

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

TRANSMETA CORPORATION,

Plaintiff and Counterclaim Defendant,

v.

INTEL CORPORATION,

Defendant and Counterclaim Plaintiff.

Civil Action No. 06-633-GMS

DEMAND FOR JURY TRIAL

**INTEL CORPORATION'S ANSWER, AFFIRMATIVE DEFENSES, AND
COUNTERCLAIMS TO TRANSMETA'S FIRST AMENDED COMPLAINT**

Defendant Intel Corporation ("Intel"), by and through its undersigned counsel, hereby demands a trial by jury on all issues so triable, and responds to the First Amended Complaint of Plaintiff Transmeta Corporation ("Transmeta") by admitting, denying, alleging, and counterclaiming as follows:

I. ANSWER

1. Intel admits that this action purports to arise under the Patent Laws of the United States, Title 35 of the United States Code. Intel further admits that this Court has subject matter jurisdiction over the allegations as plead under 28 U.S.C. §§ 1331 and 1338(a). Intel denies all remaining allegations of paragraph 1 of the Amended Complaint.

2. Intel admits that venue is proper in this District pursuant to 28 U.S.C. §§ 1391(b) and (c) and 1400(b).

3. On information and belief, Intel admits that Transmeta is a corporation organized under the laws of Delaware with its principal place of business at 3990 Freedom

Circle, Santa Clara, California 95054. Intel disputes Transmeta's characterization of its business and, therefore, denies all remaining allegations of paragraph 3 of the Amended Complaint.

4. Intel admits that it is a corporation organized under the laws of Delaware with its principal place of business at 2200 Mission College Blvd., Santa Clara, California 95052. Intel further admits that it has manufactured and sold processors within the following architectural families: P6, Pentium[®] 4, Pentium[®] M, Core[™], and Core[™] 2. Intel disputes Transmeta's characterization of Intel's business and, therefore, denies all remaining allegations of paragraph 4 of the Amended Complaint.

5. Intel admits that U.S. Patent No. 7,100,061 ("the '061 patent") is entitled "Adaptive Power Control." Intel further admits that the '061 patent states on its face that it issued on August 29, 2006, and identifies Transmeta as the assignee of the patent. Intel denies that the '061 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta's purported rights and interest in the '061 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 5 of the Amended Complaint.

6. Intel admits that U.S. Patent No. 5,895,503 ("the '503 patent") is entitled "Address Translation Method and Mechanism Using Physical Address Information Including During a Segmentation Process." Intel further admits that the '503 patent states on its face that it issued on April 20, 1999. Intel denies that the '503 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta's purported rights and interest in the '503 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 6 of the Amended Complaint.

7. Intel admits that U.S. Patent No. 6,226,733 (“the ’733 patent”) is entitled “Address Translation Mechanism and Method in a Computer System.” Intel further admits that the ’733 patent states on its face that it issued on May 1, 2001. Intel denies that the ’733 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta’s purported rights and interest in the ’733 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 7 of the Amended Complaint.

8. Intel admits that U.S. Patent No. 6,430,668 (“the ’668 patent”) is entitled “Speculative Address Translation for Processor Using Segmentation and Optical Paging.” Intel further admits that the ’668 patent states on its face that it issued on August 6, 2002, and identifies Transmeta as the assignee of the patent. Intel denies that the ’668 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta’s purported rights and interest in the ’668 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 8 of the Amended Complaint.

9. Intel admits that U.S. Patent No. 6,813,699 (“the ’699 patent”) is entitled “Speculative Address Translation for Processor Using Segmentation and Optional Paging.” Intel further admits that the ’699 patent states on its face that it issued on November 2, 2004, and identifies Transmeta as the assignee of the patent. Intel denies that the ’699 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta’s purported rights and interest in the ’699 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 9 of the Amended Complaint.

10. Intel admits that U.S. Patent No. 5,493,687 (“the ’687 patent”) is entitled “RISC Microprocessor Architecture Implementing Multiple Typed Register Sets.” Intel further admits that the ’687 patent states on its face that it issued on February 20, 1996. Intel denies that the ’687 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta’s purported rights and interest in the ’687 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 10 of the Amended Complaint.

11. Intel admits that U.S. Patent No. 5,838,986 (“the ’986 patent”) is entitled “RISC Microprocessor Architecture Implementing Multiple Typed Register Sets.” Intel further admits that the ’986 patent states on its face that it issued on November 17, 1998. Intel denies that the ’986 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta’s purported rights and interest in the ’986 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 11 of the Amended Complaint.

12. Intel admits that U.S. Patent No. 6,044,449 (“the ’449 patent”) is entitled “RISC Microprocessor Architecture Implementing Multiple Typed Register Sets.” Intel further admits that the ’449 patent states on its face that it issued on March 28, 2000. Intel denies that the ’449 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta’s purported rights and interest in the ’449 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 12 of the Amended Complaint.

13. Intel admits that U.S. Patent No. 5,737,624 (“the ’624 patent”) is entitled “Superscalar RISC Instruction Scheduling.” Intel further admits that the ’624 patent states on its

face that it issued on April 7, 1998. Intel denies that the '624 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta's purported rights and interest in the '624 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 13 of the Amended Complaint.

14. Intel admits that U.S. Patent No. 5,974,526 ("the '526 patent") is entitled "Superscalar RISC Instruction Scheduling." Intel further admits that the '526 patent states on its face that it issued on October 26, 1999. Intel denies that the '526 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta's purported rights and interest in the '526 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 14 of the Amended Complaint.

15. Intel admits that U.S. Patent No. 6,289,433 ("the '433 patent") is entitled "Superscalar RISC Instruction Scheduling." Intel further admits that the '433 patent states on its face that it issued on September 11, 2001, and identifies Transmeta as the assignee of the patent. Intel denies that the '433 patent was duly and legally issued. Intel is without knowledge or information sufficient to form a belief as to the truth of the allegations concerning Transmeta's purported rights and interest in the '433 patent and, therefore, denies those allegations. Intel denies all remaining allegations of paragraph 15 of the Amended Complaint.

16. Intel denies the allegations of paragraph 16 of the Amended Complaint.

17. Intel denies the allegations of paragraph 17 of the Amended Complaint.

18. Intel denies the allegations of paragraph 18 of the Amended Complaint.

19. Intel denies the allegations of paragraph 19 of the Amended Complaint.

20. Intel denies the allegations of paragraph 20 of the Amended Complaint.

21. Intel denies the allegations of paragraph 21 of the Amended Complaint.

22. Intel denies the allegations of paragraph 22 of the Amended Complaint.

Intel denies that Transmeta is entitled to any of the relief sought in Transmeta's prayer for relief, including that requested in paragraphs A through H.

II. DEFENSES

23. In addition to the defenses described below, Intel expressly reserves the right to allege additional defenses as they become known through the course of discovery.

FIRST DEFENSE – NON-INFRINGEMENT

24. Intel has not infringed and is not infringing, directly or indirectly, any valid claims of the '061, '503, '733, '668, '699, '687, '986, '449, '624, '526, or '433 patents.

SECOND DEFENSE – INVALIDITY

25. The '061, '503, '733, '668, '699, '687, '986, '449, '624, '526, and '433 patents are invalid for failure to meet the conditions of patentability of 35 U.S.C. § 101 et seq., including, without limitation, §§ 102, 103, and 112.

THIRD DEFENSE – OBVIOUSNESS-TYPE DOUBLE PATENTING

26. The '687, '986, '449, '624, '526, and '433 patents are invalid under the judicially created doctrine of obviousness-type double patenting.

FOURTH DEFENSE – PROSECUTION HISTORY ESTOPPEL

27. Transmeta is estopped from asserting that Intel has infringed or is infringing, directly or indirectly, one or more claims of the '061, '503, '733, '668, '699, '687, '986, '449, '624, '526, and '433 patents based on admissions, amendments, arguments, and other

statements made to the U.S. Patent and Trademark Office during prosecution of the applications leading to, or related to, the issuance of one or more of these patents.

FIFTH DEFENSE – LACHES

28. The relief sought by Transmeta based on Intel's alleged infringement of each of the '503, '733, '668, '699, '687, '986, '449, '624, '526, and '433 patents is barred in whole or in part by the doctrine of laches.

SIXTH DEFENSE – LICENSE

29. The relief sought by Transmeta based on Intel's alleged infringement of each of the '687, '986, '449, '624, '526, and '433 patents is barred in whole or in part because Intel is licensed to practice these patents either expressly or impliedly.

SEVENTH DEFENSE – EQUITABLE ESTOPPEL

30. The relief sought by Transmeta based on Intel's alleged infringement of each of the '687, '986, '449, '624, '526, and '433 patents is barred in whole or in part by the doctrine of equitable estoppel.

EIGHTH DEFENSE – OWNERSHIP

31. On information and belief, Transmeta is not the proper owner of the '503, '733, '668, '699, '687, '986, '449, '624, '526, and '433 patents it has asserted against Intel and/or has not acquired the right to sue for past damages and accordingly has no standing to bring suit for patent infringement or seek damages for any past infringement.

NINTH DEFENSE – MARKING

32. Transmeta is barred from recovering damages for any alleged infringement that occurred prior to the filing of the Complaint by 35 U.S.C. § 287.

**TENTH DEFENSE – INEQUITABLE CONDUCT AND INFECTIOUS
UNENFORCEABILITY**

A. The '061 Patent

33. The '061 patent is unenforceable under the doctrine of inequitable conduct. On information and belief, prior to the issuance of the '061 patent, one or more of the named inventors and/or others substantively involved in prosecuting the application leading to the '061 patent were aware of material information, but withheld, concealed and/or mischaracterized that information with the intent to deceive the U.S. Patent and Trademark Office.

34. Information withheld prior to the issuance of the '061 patent includes, without limitation: (1) U.S. Patent No. 5,727,208 entitled "Method and Apparatus for Configuration of Processor Operating Parameters," which was based on an application filed on July 3, 1995 and issued on March 10, 1998 to Brown; and (2) U.S. Patent No. 6,163,583 entitled "Dynamic Clocking Apparatus and System for Reducing Power Dissipation," which was based on an application filed on March 25, 1998 and issued on December 19, 2000 to Lin et al.

B. The '503, '733, '668, and '699 Patents

35. The '503, '733, '668, and '699 patents are unenforceable under the doctrine of inequitable conduct. On information and belief, prior to the issuance of the '503, '733, '668, and '699 patents, the named inventor and/or others substantively involved in prosecuting the applications leading to the '503, '733, '668, and '699 patents were aware of material information, but withheld, concealed and/or mischaracterized that information with the intent to deceive the U.S. Patent and Trademark Office. Under the doctrine of infectious unenforceability, the acts of inequitable conduct committed during the prosecution of the '503 patent render the '733, '668, and '699 patents unenforceable, the acts of inequitable conduct

committed during the prosecution of the '733 patent render the '668 and '699 patents unenforceable, and the acts of inequitable conduct committed during the prosecution of the '668 patent render the '699 patent unenforceable.

36. Information withheld prior to the issuance of the '503, '733, '668, and '699 patents includes, without limitation: (1) U.S. Patent No. 3,781,808 entitled "Virtual Memory System," which was based on an application filed on October 17, 1992 and issued on December 25, 1973 to Ahearn et al.; (2) U.S. Patent No. 4,128,875 entitled "Optional Virtual Memory System," which was based on an application filed on December 16, 1976 and issued on December 5, 1978 to Thurber et al.; (3) U.S. Patent No. 5,530,824 entitled "Address Translation Circuit," which was based on an application filed on April 4, 1994 and issued on June 25, 1996 to Peng et al.; (4) U.S. Patent No. 5,623,619 entitled "Linearly Addressable Microprocessor Cache," which was based on an application filed on July 24, 1995 and issued on April 22, 1997 to Witt; (5) U.S. Patent No. 5,768,575 entitled "Semi-Autonomous RISC Pipelines for Overlapped Execution of RISC-Like Instructions Within the Multiple Superscalar Execution Units of a Processor Having Distributed Pipeline Control for Speculative and Out-of-Order Execution of Complex Instructions," which was based on an application filed on March 13, 1995 and issued on June 16, 1998 to McFarland et al.; and (6) U.S. Patent No. 5,878,245 entitled "High Performance Load/Store Unit and Data Cache," which was based on an application filed on October 29, 1993 and issued on March 2, 1999 to Johnson et al.

37. In addition, information withheld prior to the issuance of the '503 and '733 patents includes, without limitation, U.S. Patent No. 5,617,554 entitled "Physical Address Size Selection and Page Size Selection in an Address Translator," which was based on an application filed on December 23, 1994 and issued on April 1, 1997 to Alpert et al.

38. In addition, prior to the issuance of the '503, '733, '668, and '699 patents, the named inventor and/or others substantively involved in prosecuting the applications leading to the '503, '733, '668, and '699 patents mischaracterized the scope of the claims and/or disclosure in the '503, '733, '668, and '699 patents in an attempt to distinguish them over U.S. Patent No. 4,400,774 entitled "Cache Addressing Arrangement in a Computer System," which was based on an application filed on February 2, 1981 and issued on August 23, 1983 to Toy.

C. The '687, '986, and '449 Patents

39. The '687, '986, and '449 patents are unenforceable under the doctrine of inequitable conduct. On information and belief, prior to the issuance of the '687, '986, and '449 patents, the named inventors and/or others substantively involved in prosecuting the applications leading to the '687, '986, and '449 patents were aware of material information, but withheld, concealed and/or mischaracterized that information with the intent to deceive the U.S. Patent and Trademark Office. Under the doctrine of infectious unenforceability, the acts of inequitable conduct committed during the prosecution of the '687 patent render the '986 and '449 patents unenforceable, and the acts of inequitable conduct committed during the prosecution of the '986 patent render the '449 patent unenforceable.

40. Information withheld prior to the issuance of the '687, '986, and '449 patents includes, without limitation: (1) the Intel i860 processor known, made, used, sold, offered for sale, in public use, patented and/or described in, *inter alia*, Margulis, "i860 Microprocessor Internal Architecture," *Microprocessors and Microsystems*, Vol. 14, No. 2, pp. 89-96 (March 1990); (2) the Intel i960 processor known, made, used, sold, offered for sale, in public use, patented and/or described in, *inter alia*, McGeady, "The i960CA SuperScalar Implementation of the 80960 Architecture," *IEEE*, pp. 232-240 (1990); (3) the Metaflow

architecture known, made, used, sold, offered for sale, in public use, patented and/or described in, *inter alia*, Popescu et al., “The Metaflow Architecture,” *IEEE Micro*, Vol. 11, No. 3, pp. 10-13, 63-73 (June 1991); (4) the Swordfish National Semiconductor processor known, made, used, sold, offered for sale, in public use, patented and/or described in, *inter alia*, Intrater et al., “A Superscalar Microprocessor,” *IEEE Proceedings of the 17th Convention of Electrical & Electronics Engineers In Israel*, pp. 267-270 (March 1991); (5) the Motorola 88000 family of processors known, made, used, sold, offered for sale, in public use, patented and/or described in, *inter alia*, Melear, “The Design of the 88000 RISC Family,” *IEEE Micro*, Vol. 9, No. 2, pp. 26-38 (April 1989); and (6) U.S. Patent No. 4,926,323 entitled “Streamlined Instruction Processor,” which was based on an application filed on March 3, 1988 and issued on May 15, 1990 to Baror et al.

41. In addition, information withheld prior to the issuance of the '687 and '986 patents includes, without limitation, the Multiflow processor known, made, used, sold, offered for sale, in public use, patented and/or described in, *inter alia*, Colwell et al., “A VLIW Architecture for a Trace Scheduling Compiler,” *Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 180-192 (October 1987).

D. The '624, '526, and '433 Patents

42. The '624, '526, and '433 patents are unenforceable under the doctrine of inequitable conduct. On information and belief, prior to the issuance of the '624, '526, and '433 patents, the named inventors and/or others substantively involved in prosecuting the applications leading to the '624, '526, and '433 patents were aware of material information, but withheld, concealed and/or mischaracterized that information with the intent to deceive the U.S. Patent and

Trademark Office. Under the doctrine of infectious unenforceability, the acts of inequitable conduct committed during the prosecution of the '624 patent render the '526 and '433 patents unenforceable, and the acts of inequitable conduct committed during the prosecution of the '526 patent render the '433 patent unenforceable.

43. Information withheld prior to the issuance of the '624, '526, and '433 patents includes, without limitation: (1) Paper #28 (Office Action dated June 19, 1996) from U.S. Application No. 08/481,146; (2) Paper #32 (Office Action dated November 18, 1996) from U.S. Application No. 08/481,146; and (3) U.S. Patent No. 5,590,295 entitled "System and Method for Register Renaming," which was based on an application filed on June 7, 1995 and issued on December 31, 1996 to Deosaran et al.

44. In addition, information withheld prior to the issuance of the '526 and '433 patents includes, without limitation: (1) U.S. Patent No. 5,826,055 entitled "System and Method for Retiring Instructions in a Superscalar Microprocessor," which was based on an application filed on June 7, 1995 and issued on October 20, 1998 to Wang et al.; and (2) U.S. Patent No. 5,809,276 entitled "System and Method for Register Renaming," which was based on an application filed on August 15, 1996 and issued on September 15, 1998 to Deosaran et al.

45. In addition, information withheld prior to the issuance of the '433 patent includes, without limitation: (1) U.S. Patent No. 6,131,157 entitled "System and Method for Retiring Approximately Simultaneously a Group of Instructions in a Superscalar Microprocessor," which was based on an application filed on January 20, 1998 and issued on October 10, 2000 to Wang et al.; (2) U.S. Patent No. 6,138,231 entitled "System and Method for Register Renaming," which was based on an application filed on April 21, 1998 and issued on October 24, 2000 to Deosaran et al.; and (3) U.S. Patent No. 6,272,617 entitled "System and

Method for Register Renaming,” which was based on an application filed on September 17, 1999 and issued on August 7, 2001 to Deosaran et al.

ELEVENTH DEFENSE – UNCLEAN HANDS

46. The relief sought by Transmeta is barred in whole or in part by the doctrine of unclean hands.

III. COUNTERCLAIMS

THE PARTIES

47. Intel is a corporation organized and existing under the laws of Delaware with its principal place of business in Santa Clara, California.

48. On information and belief, Transmeta is a corporation organized and existing under the laws of Delaware with its principal place of business in Santa Clara, California.

49. Transmeta has sold and continues to sell computer processors under the brand names Crusoe, Efficeon, and Efficeon 2, including, without limitation, Crusoe TM5900, Crusoe SE TM59E, Crusoe TM5800, Crusoe SE TM58E, Crusoe TM5700, Crusoe SE TM57E, Crusoe TM5600, Crusoe TM5500, Crusoe SE TM55E, Crusoe TM5400, Efficeon SF23J, Efficeon TM8820, Efficeon TM8800, Efficeon TM8620, Efficeon TM8600, and Efficeon TM8300.

JURISDICTION AND VENUE

50. These counterclaims arise under Title 35 of the United States Code. The Court has subject matter jurisdiction over these counterclaims pursuant to 28 U.S.C. §§ 1331, 1338(a), 2201, and 2202.

51. Personal jurisdiction and venue in this judicial district are proper under 28 U.S.C. §§ 1391(b), 1391(c) and 1400(b).

FIRST COUNTERCLAIM – INFRINGEMENT OF U.S. PATENT NO. 5,745,375

52. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

53. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code.

54. Intel is the owner of all right, title, and interest in U.S. Patent No. 5,745,375 (“the ’375 patent”), entitled “Apparatus and Method for Controlling Power Usage,” which was duly and legally issued to Intel on April 28, 1998. A copy of the ’375 patent is attached to this Answer, Affirmative Defenses and Counterclaims as Exhibit 1.

55. Intel has never licensed or permitted Transmeta to practice any of the inventions claimed in the ’375 patent.

56. On information and belief, Transmeta has directly infringed the ’375 patent by making, using, selling, offering for sale and/or importing processors into the United States, including this district, that infringe one or more of the claims of the ’375 patent, either literally and/or by equivalents, and/or has indirectly infringed the ’375 patent by actively inducing others to infringe the ’375 patent and/or contributing to the infringement of the ’375 patent by actively and knowingly aiding and abetting others to make, use, sell, offer for sale and/or import processors into the United States, including this district, that infringe one or more of the claims of the ’375 patent, either literally and/or by equivalents. These processors include at least one or more processors sold under the brand names Crusoe, Efficeon, and Efficeon 2. On information and belief, Transmeta continues to engage in such acts of infringement.

57. Transmeta's infringement of the '375 patent has been willful, and any further infringement of the '375 patent by Transmeta would be with full knowledge of Intel's legal interest in the '375 patent and would be deliberate and willful.

SECOND COUNTERCLAIM – INFRINGEMENT OF U.S. PATENT NO. 5,617,554

58. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

59. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code.

60. Intel is the owner of all right, title, and interest in U.S. Patent No. 5,617,554 ("the '554 patent"), entitled "Physical Address Size Selection and Page Size Selection in an Address Translator," which was duly and legally issued to Intel on April 1, 1997. A copy of the '554 patent is attached to this Answer, Affirmative Defenses and Counterclaims as Exhibit 2.

61. Intel has never licensed or permitted Transmeta to practice any of the inventions claimed in the '554 patent.

62. On information and belief, Transmeta has directly infringed the '554 patent by making, using, selling, offering for sale and/or importing processors into the United States, including this district, that infringe one or more of the claims of the '554 patent, either literally and/or by equivalents, and/or has indirectly infringed the '554 patent by actively inducing others to infringe the '554 patent and/or contributing to the infringement of the '554 patent by actively and knowingly aiding and abetting others to make, use, sell, offer for sale and/or import processors into the United States, including this district, that infringe one or more of the claims of the '554 patent, either literally and/or by equivalents. These processors include

at least one or more processors sold under the brand names Crusoe, Efficeon, and Efficeon 2. On information and belief, Transmeta continues to engage in such acts of infringement.

63. Transmeta's infringement of the '554 patent has been willful, and any further infringement of the '554 patent by Transmeta would be with full knowledge of Intel's legal interest in the '554 patent and would be deliberate and willful.

THIRD COUNTERCLAIM – INFRINGEMENT OF U.S. PATENT NO. 5,802,605

64. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

65. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code.

66. Intel is the owner of all right, title, and interest in U.S. Patent No. 5,802,605 ("the '605 patent"), entitled "Physical Address Size Selection and Page Size Selection in an Address Translator," which was duly and legally issued to Intel on September 1, 1998. A copy of the '605 patent is attached to this Answer, Affirmative Defenses and Counterclaims as Exhibit 3.

67. Intel has never licensed or permitted Transmeta to practice any of the inventions claimed in the '605 patent.

68. On information and belief, Transmeta has directly infringed the '605 patent by making, using, selling, offering for sale and/or importing processors into the United States, including this district, that infringe one or more of the claims of the '605 patent, either literally and/or by equivalents, and/or has indirectly infringed the '605 patent by actively inducing others to infringe the '605 patent and/or contributing to the infringement of the '605 patent by actively and knowingly aiding and abetting others to make, use, sell, offer for sale

and/or import processors into the United States, including this district, that infringe one or more of the claims of the '605 patent, either literally and/or by equivalents. These processors include at least one or more processors sold under the brand names Crusoe, Efficeon, and Efficeon 2. On information and belief, Transmeta continues to engage in such acts of infringement.

69. Transmeta's infringement of the '605 patent has been willful, and any further infringement of the '605 patent by Transmeta would be with full knowledge of Intel's legal interest in the '605 patent and would be deliberate and willful.

FOURTH COUNTERCLAIM – INFRINGEMENT OF U.S. PATENT NO. 5,819,101

70. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

71. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code.

72. Intel is the owner of all right, title, and interest in U.S. Patent No. 5,819,101 ("the '101 patent"), entitled "Method for Packing a Plurality of Packed Data Elements in Response to a Pack Instruction," which was duly and legally issued to Intel on October 6, 1998. A copy of the '101 patent is attached to this Answer, Affirmative Defenses and Counterclaims as Exhibit 4.

73. Intel has never licensed or permitted Transmeta to practice any of the inventions claimed in the '101 patent.

74. On information and belief, Transmeta has directly infringed the '101 patent by making, using, selling, offering for sale and/or importing processors into the United States, including this district, that infringe one or more of the claims of the '101 patent, either

literally and/or by equivalents, and/or has indirectly infringed the '101 patent by actively inducing others to infringe the '101 patent and/or contributing to the infringement of the '101 patent by actively and knowingly aiding and abetting others to make, use, sell, offer for sale and/or import processors into the United States, including this district, that infringe one or more of the claims of the '101 patent, either literally and/or by equivalents. These processors include at least one or more processors sold under the brand names Crusoe, Efficeon, and Efficeon 2. On information and belief, Transmeta continues to engage in such acts of infringement.

75. Transmeta's infringement of the '101 patent has been willful, and any further infringement of the '101 patent by Transmeta would be with full knowledge of Intel's legal interest in the '101 patent and would be deliberate and willful.

FIFTH COUNTERCLAIM – INFRINGEMENT OF U.S. PATENT NO. 5,881,275

76. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

77. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code.

78. Intel is the owner of all right, title, and interest in U.S. Patent No. 5,881,275 ("the '275 patent"), entitled "Method for Unpacking a Plurality of Packed Data into a Result Packed Data," which was duly and legally issued to Intel on March 9, 1999. A copy of the '275 patent is attached to this Answer, Affirmative Defenses and Counterclaims as Exhibit 5.

79. Intel has never licensed or permitted Transmeta to practice any of the inventions claimed in the '275 patent.

80. On information and belief, Transmeta has directly infringed the '275 patent by making, using, selling, offering for sale and/or importing processors into the United

States, including this district, that infringe one or more of the claims of the '275 patent, either literally and/or by equivalents, and/or has indirectly infringed the '275 patent by actively inducing others to infringe the '275 patent and/or contributing to the infringement of the '275 patent by actively and knowingly aiding and abetting others to make, use, sell, offer for sale and/or import processors into the United States, including this district, that infringe one or more of the claims of the '275 patent, either literally and/or by equivalents. These processors include at least one or more processors sold under the brand names Crusoe, Efficeon, and Efficeon 2. On information and belief, Transmeta continues to engage in such acts of infringement.

81. Transmeta's infringement of the '275 patent has been willful, and any further infringement of the '275 patent by Transmeta would be with full knowledge of Intel's legal interest in the '275 patent and would be deliberate and willful.

SIXTH COUNTERCLAIM – INFRINGEMENT OF U.S. PATENT NO. 6,385,634

82. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

83. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code.

84. Intel is the owner of all right, title, and interest in U.S. Patent No. 6,385,634 ("the '634 patent"), entitled "Method for Performing Multiply-Add Operations on Packed Data," which was duly and legally issued to Intel on May 7, 2002. A copy of the '634 patent is attached to this Answer, Affirmative Defenses and Counterclaims as Exhibit 6.

85. Intel has never licensed or permitted Transmeta to practice any of the inventions claimed in the '634 patent.

86. On information and belief, Transmeta has directly infringed the '634 patent by making, using, selling, offering for sale and/or importing processors into the United States, including this district, that infringe one or more of the claims of the '634 patent, either literally and/or by equivalents, and/or has indirectly infringed the '634 patent by actively inducing others to infringe the '634 patent and/or contributing to the infringement of the '634 patent by actively and knowingly aiding and abetting others to make, use, sell, offer for sale and/or import processors into the United States, including this district, that infringe one or more of the claims of the '634 patent, either literally and/or by equivalents. These processors include at least one or more processors sold under the brand names Crusoe, Efficeon, and Efficeon 2. On information and belief, Transmeta continues to engage in such acts of infringement.

87. Transmeta's infringement of the '634 patent has been willful, and any further infringement of the '634 patent by Transmeta would be with full knowledge of Intel's legal interest in the '634 patent and would be deliberate and willful.

SEVENTH COUNTERCLAIM – INFRINGEMENT OF U.S. PATENT NO. 6,418,529

88. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

89. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code.

90. Intel is the owner of all right, title, and interest in U.S. Patent No. 6,418,529 ("the '529 patent"), entitled "Apparatus and Method for Performing Intra-Add Operation," which was duly and legally issued to Intel on July 9, 2002. A copy of the '529 patent is attached to this Answer, Affirmative Defenses and Counterclaims as Exhibit 7.

91. Intel has never licensed or permitted Transmeta to practice any of the inventions claimed in the '529 patent.

92. On information and belief, Transmeta has directly infringed the '529 patent by making, using, selling, offering for sale and/or importing processors into the United States, including this district, that infringe one or more of the claims of the '529 patent, either literally and/or by equivalents, and/or has indirectly infringed the '529 patent by actively inducing others to infringe the '529 patent and/or contributing to the infringement of the '529 patent by actively and knowingly aiding and abetting others to make, use, sell, offer for sale and/or import processors into the United States, including this district, that infringe one or more of the claims of the '529 patent, either literally and/or by equivalents. These processors include at least one or more processors sold under the brand name Efficeon 2. On information and belief, Transmeta continues to engage in such acts of infringement.

93. Transmeta's infringement of the '529 patent has been willful, and any further infringement of the '529 patent by Transmeta would be with full knowledge of Intel's legal interest in the '529 patent and would be deliberate and willful.

EIGHTH COUNTERCLAIM – DECLARATORY JUDGMENT

'061 PATENT

94. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

95. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code, and the Declaratory Judgments Act, 28 U.S.C. §§ 2201 and 2202.

96. In its Amended Complaint, Transmeta alleges that Intel has infringed and is currently infringing the '061 patent by making, using, selling, offering to sell and/or importing “processors in the Pentium 4, Pentium M, Core and Core 2 families.”

97. An actual controversy exists between Transmeta and Intel by virtue of the allegations of Transmeta’s Amended Complaint in this action and Intel’s Answer as to the validity, enforceability and infringement of the '061 patent.

98. The '061 patent is invalid, unenforceable, and not infringed, as set forth in paragraphs 24 through 46 above.

99. Intel is entitled to judgment that the '061 patent is invalid, unenforceable, and not infringed.

NINTH COUNTERCLAIM – DECLARATORY JUDGMENT

'503, '733, '668, AND '699 PATENTS

100. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

101. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code, and the Declaratory Judgments Act, 28 U.S.C. §§ 2201 and 2202.

102. In its Amended Complaint, Transmeta alleges that Intel has infringed and is currently infringing the '503, '733, '668, and '699 patents by making, using, selling, offering to sell and/or importing “processors in the Pentium 4 family.”

103. An actual controversy exists between Transmeta and Intel by virtue of the allegations of Transmeta’s Amended Complaint in this action and Intel’s Answer as to the validity, enforceability and infringement of the '503, '733, '668, and '699 patents.

104. The '503, '733, '668, and '699 patents are invalid, unenforceable, and not infringed, as set forth in paragraphs 24 through 46 above.

105. Intel is entitled to judgment that the '503, '733, '668, and '699 patents are invalid, unenforceable, and not infringed.

TENTH COUNTERCLAIM – DECLARATORY JUDGMENT

'687, '986, AND '449 PATENTS

106. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

107. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code, and the Declaratory Judgments Act, 28 U.S.C. §§ 2201 and 2202.

108. In its Amended Complaint, Transmeta alleges that Intel has infringed and is currently infringing the '687, '986, and '449 patents by making, using, selling, offering to sell and/or importing “processors in the P6, Pentium 4, Pentium M, Core and Core 2 families.”

109. An actual controversy exists between Transmeta and Intel by virtue of the allegations of Transmeta's Amended Complaint in this action and Intel's Answer as to the validity, enforceability and infringement of the '687, '986, and '449 patents.

110. The '687, '986, and '449 patents are invalid, unenforceable, and not infringed, as set forth in paragraphs 24 through 46 above.

111. Intel is entitled to judgment that the '687, '986, and '449 patents are invalid, unenforceable, and not infringed.

ELEVENTH COUNTERCLAIM – DECLARATORY JUDGMENT

'624, '526, and '433 PATENTS

112. Intel incorporates herein by reference the allegations of paragraphs 1-51 of this Answer, Affirmative Defenses and Counterclaims.

113. Intel counterclaims against Transmeta pursuant to the patent laws of the United States, Title 35 of the United States Code, and the Declaratory Judgments Act, 28 U.S.C. §§ 2201 and 2202.

114. In its Amended Complaint, Transmeta alleges that Intel has infringed and is currently infringing the '624, '526, and '433 patents by making, using, selling, offering to sell and/or importing “processors in the P6, Pentium M, Core and Core 2 families.”

115. An actual controversy exists between Transmeta and Intel by virtue of the allegations of Transmeta's Amended Complaint in this action and Intel's Answer as to the validity, enforceability and infringement of the '624, '526, and '433 patents.

116. The '624, '526, and '433 patents are invalid, unenforceable, and not infringed, as set forth in paragraphs 24 through 46 above.

117. Intel is entitled to judgment that the '624, '526, and '433 patents are invalid, unenforceable, and not infringed.

DEMAND FOR JURY TRIAL

118. Pursuant to Rule 38 of the Federal Rules of Civil Procedure, Intel hereby demands a trial by jury of all issues.

PRAYER FOR RELIEF

WHEREFORE, Intel prays for judgment as follows on Transmeta's Amended Complaint and on Intel's Counterclaims:

- A. That Transmeta be adjudged to have infringed the '375, '554, '605, '101, '275, '634 and '529 patents;
- B. That Transmeta, its officers, agents, servants, employees, attorneys, and those persons in active concert or participation with any of them, be preliminarily and permanently enjoined from directly or indirectly infringing the '375, '554, '605, '101, '275, '634 and '529 patents;
- C. An accounting for damages by virtue of Transmeta's infringement of the '375, '554, '605, '101, '275, '634 and '529 patents;
- D. An award of damages to compensate Intel for Transmeta's infringement, pursuant to 35 U.S.C. § 284, said damages to be trebled because of Transmeta's willful infringement;
- E. That Transmeta's Amended Complaint be dismissed with prejudice, and that Transmeta take nothing against Intel by the Amended Complaint;
- F. For entry of an Order declaring the '061, '503, '733, '668, '699, '687, '986, '449, '624, '526, and '433 patents invalid, unenforceable, and not infringed by Intel;
- G. An assessment of pre-judgment and post-judgment interest and costs against Transmeta, together with an award of such interest and costs;
- H. That Transmeta be required to pay Intel's attorneys' fees pursuant to 35 U.S.C. § 285;

- I. That this case be declared an exceptional case; and
- J. That Intel be awarded such other and further relief as the court may deem just and proper.

YOUNG CONAWAY STARGATT
& TAYLOR, LLP

/s/ Karen E. Keller

Josy W. Ingersoll (No. 1088)

John W. Shaw (No. 3362)

Karen E. Keller (No. 4489)

The Brandywine Building

1000 West Street, 17th Floor

Wilmington, DE 19801

(302) 571-6600

kkeller@ycst.com

Attorneys for Defendant Intel Corporation

OF COUNSEL:

Matthew D. Powers

Jared Bobrow

Steven S. Cherensky

WEIL, GOTSHAL & MANGES LLP

201 Redwood Shores Parkway

Redwood Shores, CA 94065

(650) 802-3000

Kevin Kudlac

WEIL, GOTSHAL & MANGES LLP

8911 Capital of Texas Highway, Suite 1350

Austin, TX 78759

(512) 349-1930

January 9, 2007

CERTIFICATE OF SERVICE

I, Karen E. Keller, hereby certify that on January 9, 2007, I caused to be electronically filed a true and correct copy of the foregoing document with the Clerk of the Court using CM/ECF, which will send notification that such filing is available for viewing and downloading to the following counsel of record:

Jack B. Blumenfeld, Esquire
Karen Jacobs Loudon, Esquire
Morris Nichols Arsht & Tunnell
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899-1347

I further certify that on January 9, 20007, I caused a copy of the foregoing document to be served by hand delivery on the above-listed counsel of record and on the following in the manner indicated:

BY E-MAIL

Robert C. Morgan, Esquire
Laurence S. Rogers, Esquire
Steven Pepe, Esquire
ROPES & GRAY LLP
1251 Avenue of the Americas
New York, NY 10020

Norman H. Beamer, Esquire
ROPES & GRAY LLP
525 University Avenue
Palo Alto, CA 94301

YOUNG CONAWAY STARGATT & TAYLOR, LLP

/s/ Karen E. Keller

Karen E. Keller (No. 4489)
The Brandywine Building
1000 West Street, 17th Floor
Wilmington, Delaware 19801
(302) 571-6600
kkeller@ycst.com

Attorneys for Intel Corporation

EXHIBIT 1



US005745375A

United States Patent [19]**Reinhardt et al.**[11] **Patent Number:** **5,745,375**[45] **Date of Patent:** **Apr. 28, 1998**[54] **APPARATUS AND METHOD FOR CONTROLLING POWER USAGE**

[75] Inventors: **Dennis Reinhardt**, Palo Alto; **Ketan Bhat**, Mountain View; **Robert T. Jackson**; **Borys Senyk**, both of San Jose; **Eugene P. Matter**; **Stephen H. Gunther**, both of Folsom, all of Calif.

[73] Assignee: **Intel Corporation**, Santa Clara, Calif.

[21] Appl. No.: **537,146**

[22] Filed: **Sep. 29, 1995**

[51] Int. Cl.⁶ **H02J 03/16**

[52] U.S. Cl. **364/492; 395/750**

[58] Field of Search **364/492, 707; 395/750; 136/290, 293; 320/30, 43**

[56] **References Cited****U.S. PATENT DOCUMENTS**

4,238,784 12/1980 Keen et al. 341/126

5,254,992 10/1993 Keen et al. 341/119

Primary Examiner—Ellis B. Ramirez

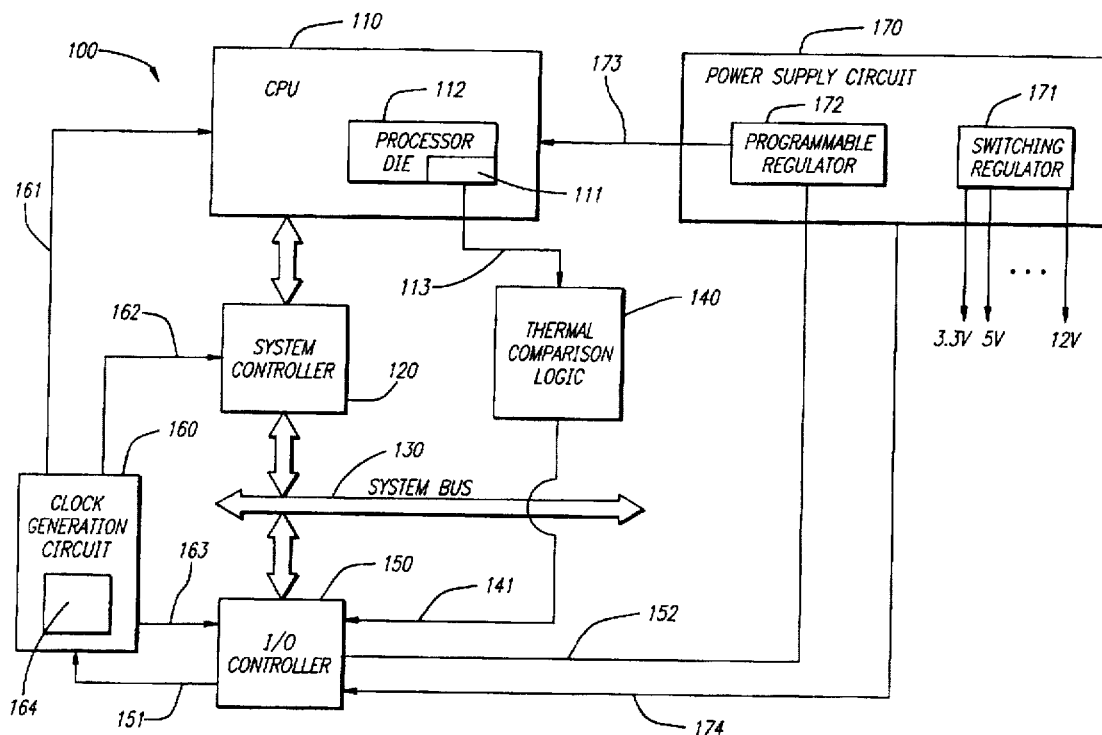
Assistant Examiner—Thomas Peeso

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A power control circuit and corresponding technique for reducing power consumption by an electronic device and thereby increasing performance. The power control circuit comprises a controller, a clock generation circuit and a power supply circuit. The controller detects whether a condition exists to scale the voltage and frequency of the electronic device and in response, signals the clock generation circuit to perform frequency scaling on the electronic device and the power supply circuit to perform voltage scaling on the electronic device. The condition may include a situation where the temperature of the electronic device is detected to have exceeded a thermal band. The condition may also include a situation where the electronic device is detected to be idle for a selected percentage of its run time.

33 Claims, 4 Drawing Sheets

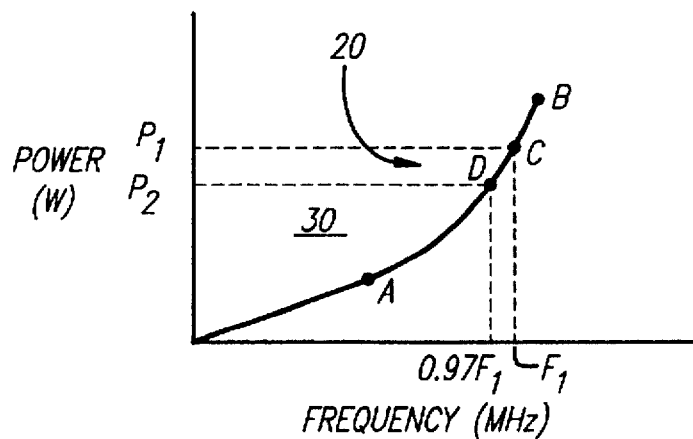
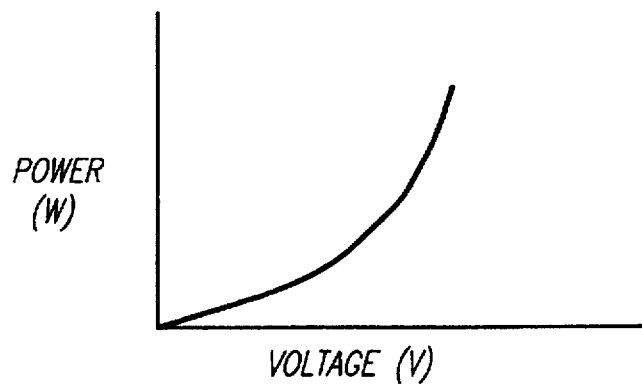
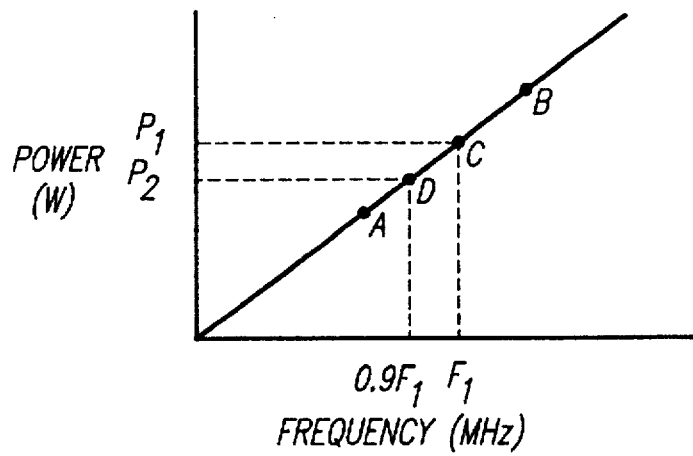


U.S. Patent

Apr. 28, 1998

Sheet 1 of 4

5,745,375



U.S. Patent

Apr. 28, 1998

Sheet 2 of 4

5,745,375

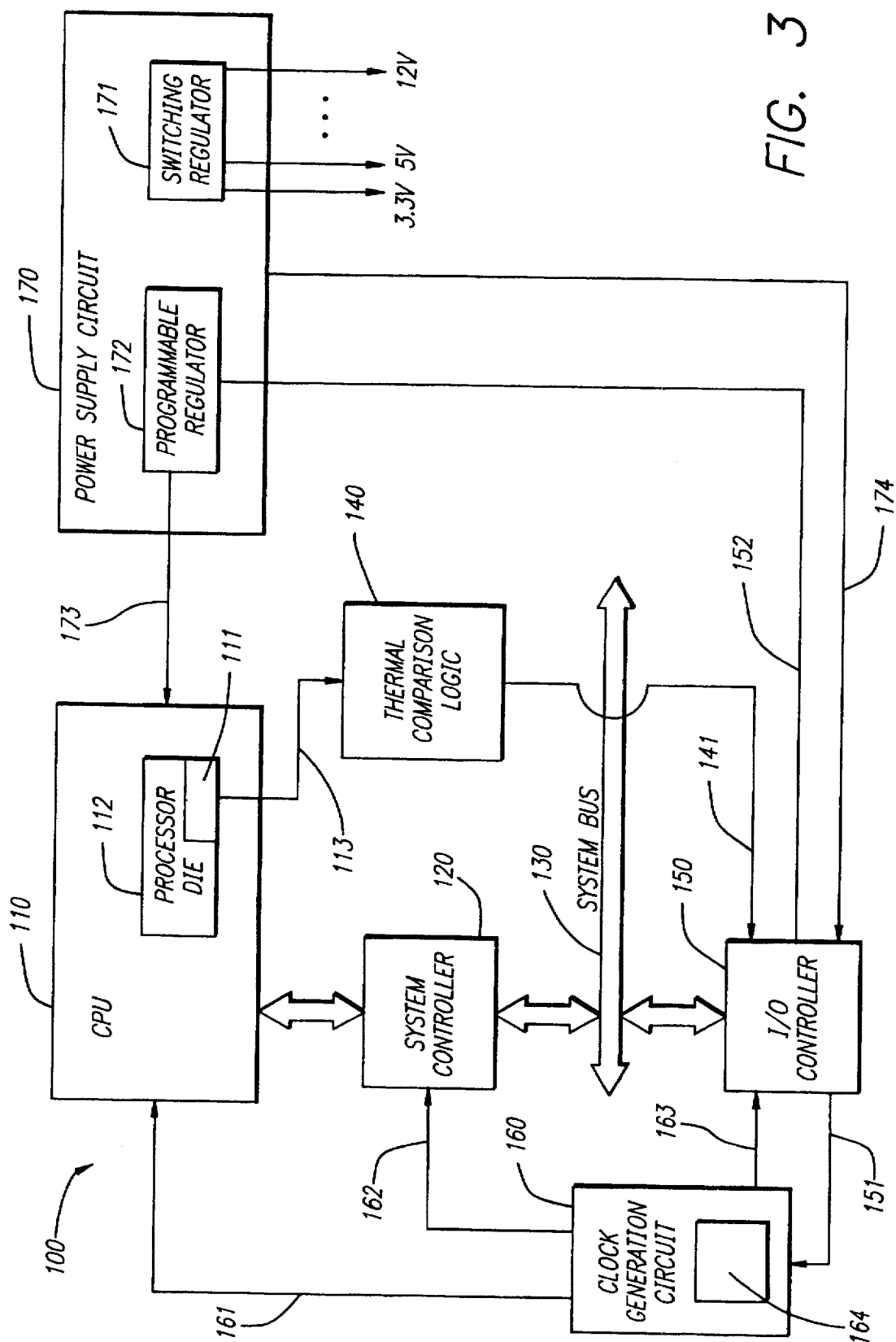


FIG. 3

U.S. Patent

Apr. 28, 1998

Sheet 3 of 4

5,745,375

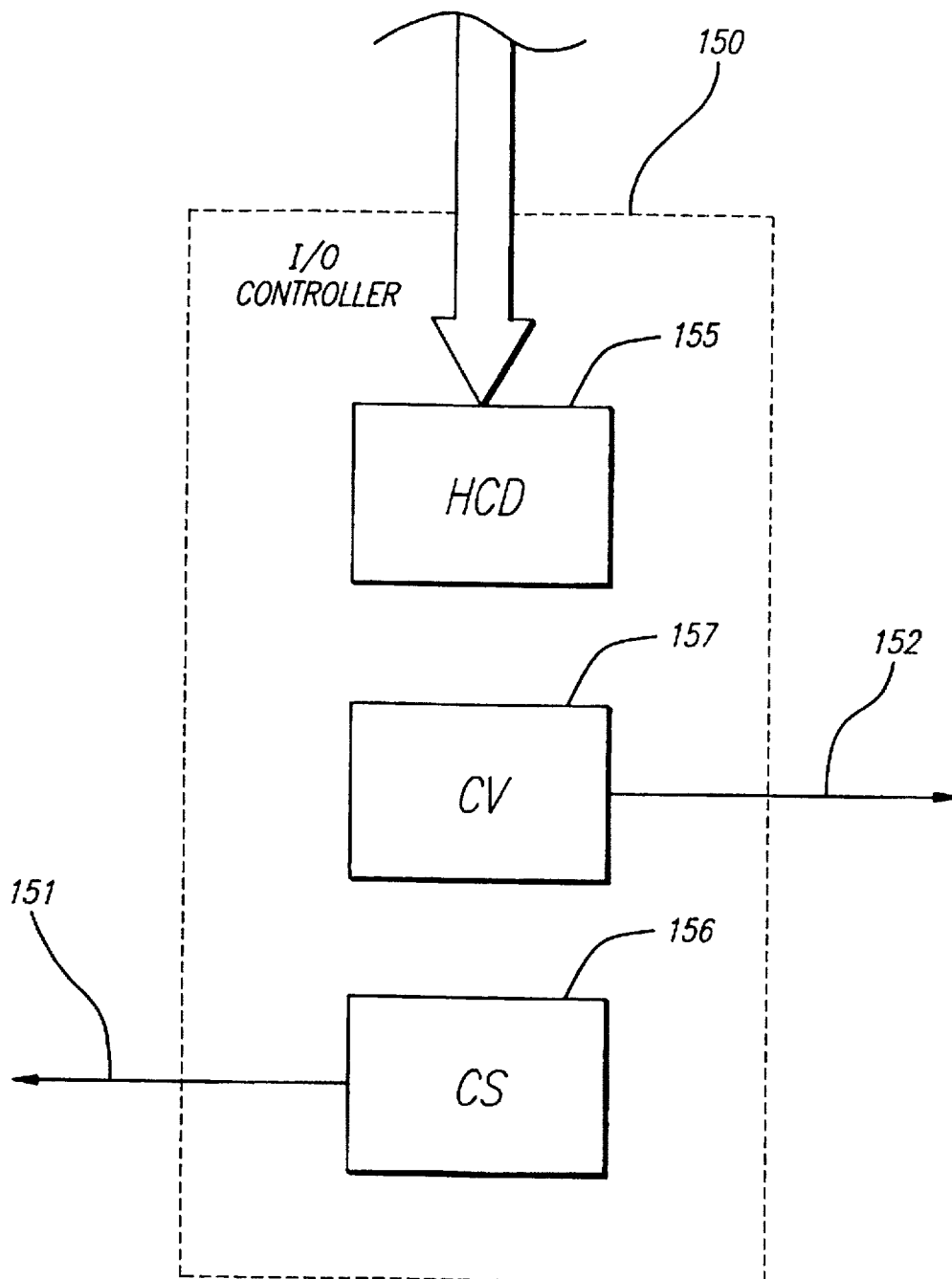


FIG. 4

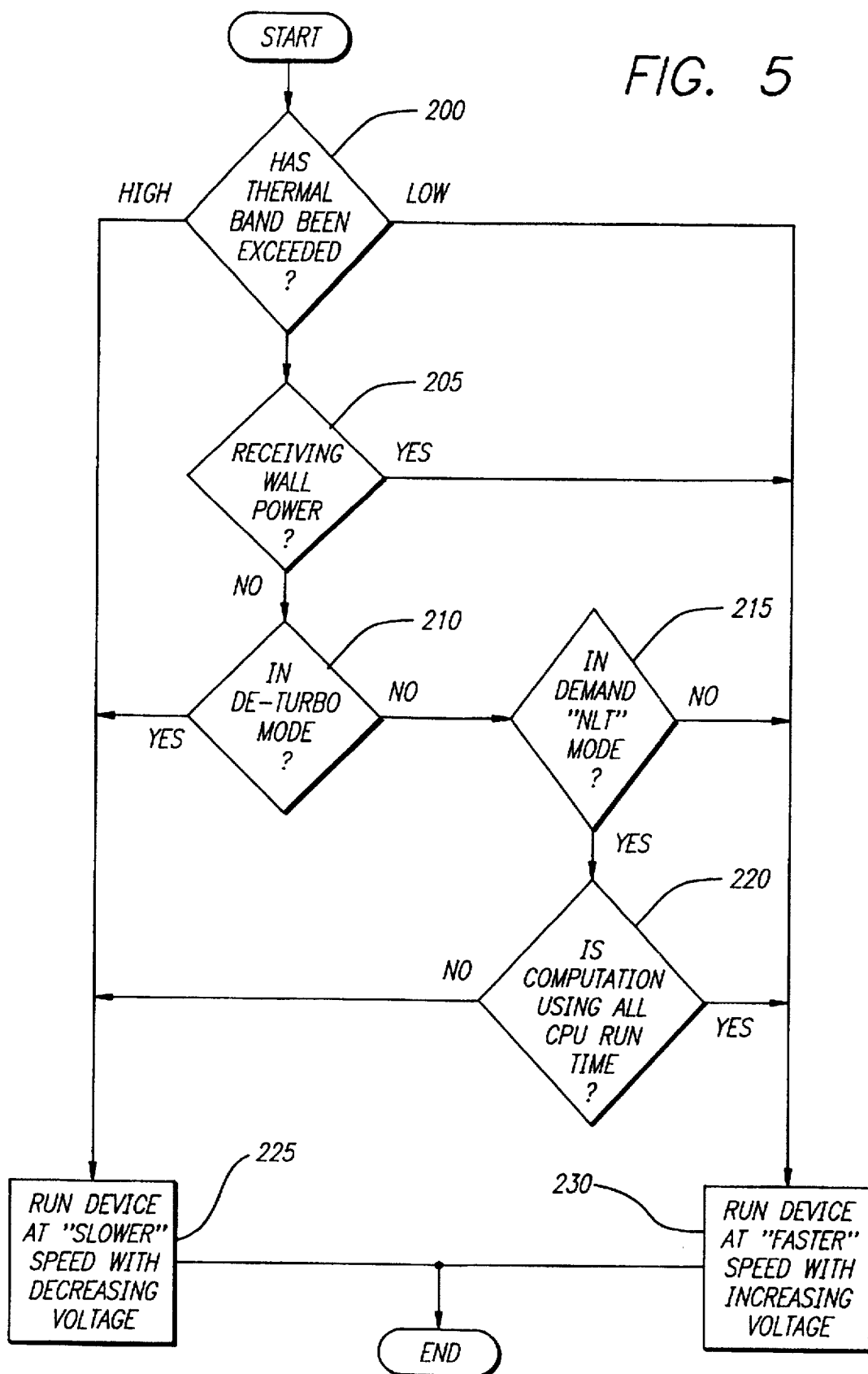
U.S. Patent

Apr. 28, 1998

Sheet 4 of 4

5,745,375

FIG. 5



5,745,375

1

APPARATUS AND METHOD FOR CONTROLLING POWER USAGE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of electronic devices. More particularly, the present invention relates to reducing power consumption by an electronic device through voltage and frequency scaling.

2. Description of Art Related to the Invention

Over the last few years, there have been many advances in semiconductor technology which have resulted in the development of improved electronic devices having integrated circuits operating at higher frequencies and supporting additional and/or enhanced features. While these advances have enabled hardware manufacturers to design and build faster and more sophisticated hardware products (e.g., computers, peripheral devices, etc.), they have also imposed a disadvantage primarily experienced by battery-powered laptop or notebook style computers. In particular, these improved electronic devices consume more power and dissipate more heat as a by-product than those past generation electronic devices.

It is well-known that modern battery-powered laptop computers place a high premium on reducing power consumption because such reduction extends its battery life. Currently, one primary technique to reduce power consumption of laptop computers is to lower the frequency of the clocking signal supplied to one of its electronic devices, namely its central processing unit ("CPU"). This technique (referred to herein as "frequency reduction") usually is accomplished by dividing the clocking signal supplied to the CPU (i.e., the CPU clock) or, in the alternative, halting the clocking signal for brief time intervals so that the average operating frequency is reduced.

Referring to FIG. 1, a graph illustrating power savings realized by any electronic device based on the conventional frequency reduction technique is shown. It is well-known that electronic devices, in general, are designed to operate within a specific voltage range. This frequency range 10 is represented as being between points A and B, where point A represents the minimum frequency required for the electronic device to operate and point B represents the maximum frequency that the electronic device can support. In theory, power is directly proportional to frequency as presented herein. Thus, as shown through points C and D, a reduction in the operating frequency of the electronic device by ten percent (10%) will reduce its total power consumption by ten percent (10%) from P1 to P2. Of course, true power savings are not exactly proportional to frequency reduction because most every hardware product is implemented with electronic devices which consume power but are frequency-independent (e.g., displays for computer systems).

This conventional frequency reduction technique imposes a number of disadvantages. One paramount disadvantage is that the frequency reduction offers minimal conservation of battery life because the amount of energy required by the electronic device undergoing frequency reduction to perform a certain task can remain constant. In some situations, depending on the chosen configuration between frequency-dependent and frequency independent devices within a product like a laptop computer, frequency reduction may adversely effect battery life conservation. This is largely due to the fact that the electronic device, while operating at a slower frequency, requires extra operating time to complete the task. As a result, this extra operating time causes the

2

frequency-independent devices within the product to consume more energy which, in some cases, will exceed any energy savings realized by reduced the operating frequency of the electronic device.

Hence, it is desirous to create a power control circuit and develop a technique for reducing power consumption which can be utilized by any type of electronic device, especially microprocessors, to more effectively reduce its power consumption without substantially mitigating performance.

Another advantage provided by the power control circuit is that it encourages the implementation of full-frequency, full-voltage electronic devices within power sensitive hardware products rather than configuring the electronic device to function if a worst-case condition occurs. Instead, the electronic device relies on voltage and frequency scaling to dynamically configure the electronic device based on its various conditions at the time. Thus, the overall performance of the product implementing the electronic device is enhanced.

Yet another advantage is to enable companies to reduce their inventories of electronic devices dedicated to either laptop or desktop systems by eliminating the need for electronic devices calibrated to have a specific voltage and operating frequency, but rather dynamically calibrating the voltage and frequency of the electronic device based on the current conditions experienced by the electronic device.

BRIEF SUMMARY OF THE INVENTION

The present invention relates to a power control circuit and a technique for, among other things, reducing power consumption by an electronic device. The power control circuit comprises a controller, a clock generation circuit and a power supply circuit. The controller, governed by power management software, signals the power supply circuit and the clock generation circuit to perform voltage and frequency scaling on the electronic device, provided at least one of two conditions occurs. For example, a first condition is where the temperature of the electronic device is detected to have exceeded a thermal band. A second condition is where the electronic device is detected to be idle for a selected percentage of power-on time.

BRIEF DESCRIPTION OF THE DRAWINGS

The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

FIG. 1 is an illustrative diagram of theoretical power savings realized by a conventional frequency reduction technique.

FIG. 2a is an illustrative diagram of the theoretical "squared" relationship between voltage and power.

FIG. 2b is an illustrative diagram of power savings realized by an electronic device which is controlled through voltage and frequency scaling as provided by the present invention.

FIG. 3 is an illustrative block diagram of a power control circuit of the present invention employed within a standard computer system including an input/output ("I/O") controller, a clock generation circuit and a power supply circuit.

FIG. 4 is an illustrative block diagram of a plurality of registers employed within the I/O controller to store information to enable the clock generation circuit to perform frequency scaling and the power supply circuit to perform voltage scaling.

5,745,375

3

FIG. 5 illustrates the operations performed by the present invention to optimally reduce power consumption by the electronic device through voltage and frequency scaling.

DETAILED DESCRIPTION OF THE INVENTION

The present invention describes a system and method for controlling power consumption of at least one electronic device through both voltage and frequency scaling. The following detailed description is presented largely in terms of graphs, block diagrams and a flowchart which collectively illustrate the present invention in detail but does not discuss well-known circuits or process steps to avoid unnecessarily obscuring the present invention. The flowchart illustrates a series of steps leading to a desired result. These steps require physical manipulations of physical quantities in the form of electrical or magnetic signals capable of being stored, transferred, combined, compared or otherwise manipulated.

Referring to FIG. 2a, an illustrative graph of the relationship between voltage and power is shown. As noted in Equations 1 and 2 below, valid for many electronic devices (e.g., CMOS), theoretically, power has a "squared" law dependence with voltage which, in turn, has a generally proportional relationship with the operating frequency.

Equation 1:

Power = $C \times V^2 \times F \times \text{Act}$, where

"C"=total capacitance of the electronic device;

"V"=total voltage supplied to the electronic device;

"F"=operating frequency of the electronic device; and

"Act"=percentage of gates of the electronic devices changing state for a given clock cycle.

Equation 2:

$V \propto F$, where $V_1 \geq V \geq V_2$ and V_1 is the maximum operating voltage supported by the electronic device, and $V \propto (k \times F)$, where $V_2 \geq V \geq V_3$ where "k" is a constant less than one when voltage scaling is performed outside a voltage range defined below.

Equation 2 is a linearized approximation.

Thus, according to Equation 1, a ten percent decrease (10%) in voltage constitutes a nineteen percent (19%) decrease in power since $C \times (0.90V)^2 \times F \times \text{Act} = (0.81) \times \text{Power}$.

Referring now to FIG. 2b, an illustrative graph of the power saving realized by an electronic device by performing combined voltage and frequency scaling is shown. Similar to FIG. 1, the electronic device is operational within a voltage range 20 which is defined between point A (minimum operating voltage of the electronic device) and point B (maximum operating voltage). Furthermore, to be consistent with FIG. 1, points C and D represent the operational frequency of the electronic device at power levels P1 and P2, respectively. Thus, in the present invention, by decreasing the operational frequency and voltage of the electronic device (at point C) by slightly more than three percent (to point D), the power consumed by the electronic device is decreased by approximately ten percent (10%) since

$C \times (0.966V)^2 \times (0.966F) \times \text{Act} = (0.901) \times \text{Power}$.

Clearly, while the realized power savings is generally equivalent to that obtained through the conventional frequency reduction technique, the operating frequency of the electronic device has diminished only three percent (3%) rather than ten percent (10%). It is contemplated that voltage and frequency scaling may occur in the voltage range 20, however only frequency scaling may occur for the electronic device along a low-voltage range 30 up to point A. This is

4

due to the fact that voltage scaling in the low-voltage range 30 would cause the electronic device to become inoperative.

Referring to FIG. 3, one embodiment of a power control circuit employed within a computer system to control power consumption by an electronic device (e.g., a microprocessor) is illustrated. Although the electronic device is shown as a microprocessor because of its reputation of being one of the main power consuming chips within a computer system, the power control circuit is capable of controlling power consumption by other types of electronic devices such as controllers.

The computer system 100 comprises a central processing unit ("CPU") 110, a system controller 120, a system bus 130, thermal comparison logic 140, an input/output ("I/O") controller 150, a clock generation circuit 160 and a power supply circuit 170. After the computer system is powered-on and the user has selected a software application from main memory, mass storage memory device (e.g., IDE device) or an external disk drive to perform a certain task, the I/O controller 150 is configured by thermal management software stored within the CPU 110 to facilitate voltage and frequency scaling of the CPU 110 if at least one of two conditions occurs; namely, the temperature of the CPU 110 exceeds a thermal band or the CPU 110 is experiencing excessive idle time. The "thermal band" is represented by an absolute hardware limit (requiring immediate device shut-off if exceeded) and programmable software upper and lower limits. These software limits represent thermal limits where, if exceeded, "throttling" (i.e., decreasingly scale voltage and frequency) or "de-throttling" is recommended.

As shown, a temperature sensing component (e.g., thermistor and the like) 111 is coupled to a processor die 112 of the CPU 110 in order to monitor the temperature of the processor die 112 through thermal dissipation results, and to detect when the temperature has exceeded the thermal band. Thereafter, the temperature sensing component 111 transmits an analog signal to the thermal comparison logic 140 via control line 113. The thermal comparison logic 140 receives the analog signal and converts it into a digital signal which is input into the I/O controller 150 via a temperature control line 141. This digital signal, when asserted, indicates to the I/O controller 150 that the CPU 110 is operating at a temperature outside its thermal band. As a result, the I/O controller 150 needs to perform an operation to reduce the temperature of the processor die 112 within the CPU 110.

To reduce the temperature of the processor die 112, the I/O controller 150 programs a register 164 within the clock generation circuit 160 by propagating user-configured, programmable information stored within the I/O controller 150 into the register 164 via control line 151. The programmed information indicates how much (usually in a percentage value) the operating frequency of the clocking signal, supplied to at least the CPU 110 by the clock generation circuit 160 via clock lines 161, is to be altered. In some CPU implementations as shown, the clocking signal utilized by the system bus 130 must bear a fixed relationship with the clocking signal input into the CPU 110 (i.e., CPU clock). As a result, the clocking signals of the system controller 120 and the system bus 130 are reduced in proportion to the CPU clock. The clock generation circuit 160 monitors the value of the register 164 and appropriately modifies the frequency of the clocking signals transferred through clock lines 161-163.

After determining that the operating frequency has been reduced though any well known technique (e.g., signaling, preset delay time, etc.), the I/O controller 150 generates a voltage modification control signal to the power supply

5,745,375

5

6

circuit 170 via control line 152. The power supply circuit 170 includes a switching regulator 171 and a programmable regulator 172. Although not shown, the power supply circuit 170 includes a sensing circuit to indicate to the I/O controller 150 whether power is provided to the computer system 100 from a wall socket or from a battery source. The programmable regulator 172 receives the voltage modification control signal from the I/O controller 150 which indicates the amount of CPU core voltage, which is transferred to the processor die 112 by the programmable regulator 172 through power line 173, is reduced. The switching regulator 171, however, is unaffected by the voltage modification control signal and continues to provide power (3.3V, 5V, 12V, etc.) to power planes of the computer system 100.

Thus, in order to optimally diminish power consumption by the CPU 110 without disproportionate sacrifice of its speed, the CPU 110 first experiences frequency reduction and then voltage reduction. This order of scaling guarantees that the CPU 110 does not experience failure. However, it is contemplated that de-throttling the CPU 110 (i.e., increasing its voltage and frequency) requires the CPU core voltage to be appropriately increased before the operating frequency is increased.

The second condition, i.e., the CPU is experiencing excessive "idle" time, typically occurs when the computer system 100 is running a software application that does not require optimal performance of the CPU 110 such as, for example, various legacy software applications, word processing programs, etc. Thus, power consumption can be optimally reduced by monitoring the amount of idle time experienced by the CPU 110.

It is well known in the art that power management software such as Advanced Power Management ("APM") software, which is stored within the main memory of the computer system and operates transparently to the user, monitors whether the CPU 110 is idle or is performing useful computations. When the CPU 110 is idle, the power management software in one implementation generates a HALT instruction and causes the CPU 110 to produce a halt acknowledgment cycle. The halt acknowledgment cycle is propagated through the system controller 120 onto the system bus 130. Upon detecting that the CPU 110 is producing the halt acknowledgment cycle, the I/O controller 150 sets its halt cycle detect ("HCD") storage element as shown in FIG. 4. Thereafter, the power management software periodically scans the HCD storage element and in the event that the HCD storage element is set frequently (e.g., at least approximately five to ten percent 5%-10% of its run time), the computer system is throttled to perform voltage and frequency scaling. In such case, the I/O controller 150 performs the voltage and frequency scaling operations in the same manner as discussed above with respect to the first condition.

Some implementations do not generate a halt instruction but, instead, save power by other means (e.g., frequency scaling). In such cases, the HCD storage element 155 is modified in obvious ways to detect such other means.

Referring now to FIG. 4, an embodiment of the I/O controller 150 is shown. The I/O controller 150 includes the HCD storage element 155, a clock speed ("CS") storage element 156 and a CPU core voltage ("CCV") storage element 157. The HCD storage element 155 preferably is a single bit register indicating dynamically how frequently the CPU is in normal or idle state. More specifically, the HCD storage element 155 is set when the CPU is idle and is reset when the CPU is in its normal operating state. Thus, power management software requests the I/O controller 150 to

perform voltage and frequency scaling when the HCD storage element 155 is frequently set and return the CPU to its maximum operating frequency and corresponding voltage when the HCD storage element 155 is frequently reset.

The CS storage element 156 is configured as a "n" bit register ("n" being an arbitrary whole number) to incorporate a frequency slewing constant which is used to throttle the frequency of the CPU. This is accomplished by transferring the frequency slewing constant from the CS storage element 156 into the register 164 of the clock generation circuit. Similarly, the CCV storage element 157 is configured to incorporate a voltage slewing constant which is used to incrementally throttle the CPU core voltage provided by the power supply circuit. The voltage slewing constant is transferred into the programmable regulator of the power supply circuit as shown in FIG. 3.

Referring now to FIG. 5, an illustrative flowchart featuring the operational steps of the present invention is shown. In Step 200, the temperature of the electronic device is monitored to ascertain whether it has exceeded the thermal band. If the predetermined thermal threshold has been exceeded, the electronic device undergoes both voltage and frequency scaling to reduce its power consumption (Step 225) or if the imposed low thermal band is exceeded, the device is capable of operating at a higher voltage and frequency. If the electronic device has not exceeded its thermal band, a determination is made as to whether the electronic device is receiving alternating current ("AC") power from a conventional wall socket or is receiving direct current ("DC") power through a battery power supply (Step 205). If the electronic device is receiving power from the conventional wall socket, no voltage and frequency scaling is performed on the electronic device as shown in Step 230, provided the condition according to Step 200 is not met.

Alternatively, if the electronic device is receiving power from the battery power supply, a determination is required as to whether the user has enabled at least one of two power saving modes to reduce power consumption by the electronic device. The first power mode to be checked is whether the hardware product is in "De-turbo mode" (Step 210). In De-turbo mode, the user selectively sets (in user setup) a desired operating frequency of the electronic device to be less than the maximum operating frequency. This can be performed in laptop computers by depressing a switch located on the computer. If the hardware product employing the electronic device is in De-turbo mode, the voltage and frequency of the electronic device is appropriately scaled as configured (Step 225).

However, if that hardware product is not configured to support the De-turbo mode, a second determination is made as to whether the user has enabled a second power saving mode referred to as a "Demand Non-Linear Throttling" ("DNLT") mode (Step 215). In this mode, if enabled by the user, software will transparently alter the voltage and frequency of the electronic device based on amount of idle time experienced by the electronic device (Step 225), provided the conditions associated with Steps 200 and 210 do not indicate the contrary. If the DNLT mode is disabled, no voltage and frequency scaling is performed. Otherwise, when DNLT mode is enabled and the electronic device is frequently experiencing idle time thereby indicating that the device is being utilized to its full capability, voltage and frequency scaling is performed on the electronic device until it is operating at its maximum capability (Steps 220, 225). In the event that the DNLT mode is enabled and the electronic device is operating at its full capability, no voltage and frequency scaling is performed on the electronic device

5,745,375

7

(Steps 220, 230). This process is continued to monitor the electronic device to optimize its performance and especially reduce its power consumption.

The present invention described herein may be designed in many different embodiments evident to one skilled in the art than those described without departing from the spirit and scope of the present invention. The invention should, therefore, be measured in terms of the claims which follow.

What is claimed is:

1. A power control circuit adapted for use by an electronic device comprising:

a clock generation circuit that supplies a clock signal having a scalable frequency to the electronic device; a power supply circuit that provides a power supply signal having a scalable voltage to the electronic device; and a controller coupled to said clock generator circuit and said power supply circuit, said controller generates a first and second signal in response to an event in order to dynamically control power usage by the electronic device, the power usage is capable of being (i) reduced by decreasing the frequency of the clock signal followed by the voltage of the power supply signal or (ii) increased by increasing the voltage followed by the frequency.

2. The power control circuit according to claim 1 further comprising a thermal detection circuit that monitors a temperature of the electronic device and outputs a third signal to said controller upon detecting said event.

3. The power control circuit according to claim 2, wherein said thermal detection circuit includes

a temperature sensing device coupled to of the electronic device; and

thermal comparison logic coupled to said temperature sensing device and said controller, said thermal comparison logic receives a signal from said temperature sensing device, compares said signal to a requisite temperature level and transfers said signal into said third signal which, when asserted, indicates that the electronic device has exceeded said thermal band.

4. The power control circuit according to claim 2, wherein said event detected by said thermal detection circuit includes a condition where the electronic device has exceeded a thermal band.

5. The power control circuit according to claim 1, wherein said controller includes a clock speed storage element to contain a frequency slewing constant used to regulate adjustment of the scalable frequency and a core voltage storage element to contain a voltage slewing constant used to regulate adjustment of the scalable voltage.

6. The power control circuit according to claim 1, wherein said clock generation circuit reduces the frequency of said scalable clock signal upon receiving said first signal.

7. The power control circuit according to claim 1, wherein said controller further detects whether the electronic device is idle for at least a predetermined percentage of its run time and in response outputs said first and second signals to commence frequency and voltage scaling of the electronic device.

8. A power control circuit for an electronic device comprising:

clock means for scaling an operating frequency of a clock signal supplied to at least the electronic device;

power means for scaling a voltage supplied to the electronic device; and

control means for at least detecting whether the electronic device is idle for a predetermined percentage of a

8

run-time of said electronic device and for outputting a plurality of control signals to commence dynamic scaling of the operating frequency and voltage supplied to the electronic device when idle for at least said predetermined percentage of said run-time, said control means being coupled to said clock means and said power means.

9. The power control circuit according to claim 8, wherein said control means includes a clock speed storage element, a core voltage storage element and a halt cycle detect storage element.

10. The power control circuit according to claim 9, wherein said halt cycle detect storage element is a one-bit register.

11. The power control circuit according to claim 10, wherein said clock means reduces said operating frequency of said clock signal upon detecting said halt cycle detect storage element is set.

12. The power control circuit according to claim 8 further comprising thermal detection means for monitoring a temperature of the electronic device and for producing a signal when the electronic device has exceeded a thermal band.

13. The power control circuit according to claim 12, wherein said thermal detection means includes

sensing means for sensing said temperature of the electronic device; and

comparison means for receiving a temperature signal from said sensing means and for digitizing said temperature signal into said signal which, when asserted, indicates that the electronic device has exceeded said thermal band, said comparison means is coupled to said sensing means.

14. The power control circuit according to claim 13, wherein said clock means reduces said operating frequency of said clock signal supplied to the electronic device upon said thermal detection means asserting said signal.

15. The power control circuit according to claim 14, wherein said power supply means reduces said voltage supplied to the electronic device after said operating frequency of said clock signal has been reduced.

16. A computer system comprising:

a processor; and

a power control circuit coupled to the processor, the power control circuit including

a clock generation circuit that supplies a clock signal having a scalable frequency to said processor in response to a first signal,

a power supply circuit that provides a power supply signal having a scalable voltage to said processor in response to a second signal, and

a controller coupled to said clock generation circuit and said power supply circuit, said controller generates said first and second signal in response to an event in order to dynamically control power usage by said processor, the power usage is capable of being either incrementally increased or decreased in order to obtain a desired tradeoff between performance and power usage by the processor.

17. The computer system according to claim 16, wherein said power control circuit further comprises a thermal detection circuit that monitors a temperature of the electronic device and out-puts a third signal to said controller upon detecting said event.

18. The computer system according to claim 17, wherein said event detected by said thermal detection circuit of said power control circuit includes a condition where the electronic device has exceeded a thermal band.

5,745,375

9

19. The computer system according to claim 18, wherein said thermal detection circuit of said power control circuit includes

a temperature sensing device coupled to said processor; and

thermal comparison logic coupled to said temperature sensing device and said controller, said thermal comparison logic receives a signal from said temperature sensing device, compares said signal to ascertain whether said processor has exceeded the thermal band, and transmits said third signal in an asserted state to indicate that said processor has exceeded said thermal band.

20. The computer system according to claim 16, wherein said controller of said power control circuit includes a clock speed storage element and a core voltage storage element.

21. The computer system according to claim 16, wherein said clock generation circuit reduces the frequency of the clock signal upon receiving said first signal.

22. The computer system according to claim 21, wherein said power supply circuit reduces said scalable voltage provided through said power supply signal upon receiving said second signal.

23. The computer system according to claim 16, wherein said controller of said power control circuit further detects whether said processor is idle for at least a predetermined percentage of its run time and in response outputs said first and second signals to commence dynamic frequency and voltage scaling of said processor.

24. A computer system comprising

processor means for processing information;

bus means for transferring said information internally within the computer system;

system control means for transferring said information between said processor means and said bus means, said system control means is coupled to said processor means and said bus means; and

power control means for reducing power consumption by at least said processor, said power control means includes

clock means for scaling an operating frequency of a clock signal supplied to at least said processor means,

power means for scaling a voltage provided to said processor means, and

control means for at least detecting whether the electronic device is idle for a predetermined percentage of a run-time of said electronic device and for outputting a plurality of control signals to commence frequency and voltage scaling of the processor means when idle for at least said predetermined percentage of said run-time, said control means being coupled to said clock means and said power means.

25. The computer system according to claim 24, wherein said control means of said power control means of said power control means includes a clock speed storage element, a core voltage storage element and a halt cycle detect storage element.

26. The computer system according to claim 24, wherein said power control means further comprises thermal detection means for monitoring a temperature of said processor means and for producing a signal when said processor means has exceeded a thermal band.

27. The computer system according to claim 26, wherein said thermal detection means of said power control means includes

10

sensing means for sensing said temperature of said processor means; and

comparison means for receiving a temperature signal from said sensing means and for digitizing said temperature signal into said signal which, when asserted, indicates that said processor means has exceeded said thermal band, said comparison means is coupled to said sensing means.

28. The computer system according to claim 27, wherein said clock means reduces operating frequency of said clock signal upon said thermal detection means asserting said signal.

29. The computer system according to claim 28, wherein said power supply means reduces said voltage supplied to said processor means after said operating frequency of said clock signal.

30. A method to control power consumption by an electronic device, the method comprising the steps of:

determining whether a first condition exists which requires power consumption by the electronic device to be reduced;

scaling an operating frequency of a clocking signal supplied to the electronic device if said first condition exists; and

scaling a voltage supplied to the electronic device subsequent to scaling the operating frequency if said first condition exists.

31. The method according to claim 30, wherein said step of determining whether said first condition exists includes the step of determining whether the electronic device is operating at a temperature greater than a specific thermal band.

32. The method according to claim 31, wherein the method further comprises the step of

determining whether the electronic device is coupled to one of a battery source and a power outlet, wherein if the electronic device is coupled to said battery source,

determining whether the electronic device is experiencing at least a predetermined percentage of idle time compared to a run-time of the electronic device,

scaling said operating frequency of said clocking signal supplied to the electronic device if the electronic device is experiencing at least said predetermined percentage of idle time, and

scaling said voltage supplied to the electronic device if the electronic device is experiencing at least said predetermined percentage of idle time, and

if the electronic device is coupled to said power outlet, operating the electronic device at said operating frequency and said voltage.

33. A method to control power consumption and performance of an electronic device, the method comprising the steps of:

determining whether a first condition exists which for an increase in increased power usage by the electronic device in order to increase performance of the electronic device;

increasing a voltage supplied to the electronic device if the first condition exists; and

increasing an operating frequency of a clocking signal supplied to the electronic device after an increase in the voltage if the first condition exists.

* * * * *

EXHIBIT 2



US005617554A

United States Patent [19]

Alpert et al.

[11] Patent Number: **5,617,554**
 [45] Date of Patent: **Apr. 1, 1997**

[54] PHYSICAL ADDRESS SIZE SELECTION AND PAGE SIZE SELECTION IN AN ADDRESS TRANSLATOR

[75] Inventors: **Donald B. Alpert**, Santa Clara;
Kenneth D. Shoemaker, Saratoga, both
 of Calif.; **Kevin C. Kahn**, Portland;
Konrad K. Lai, Aloha, both of Oreg.

[73] Assignee: **Intel Corporation**, Santa Clara, Calif.

[21] Appl. No.: **372,805**

[22] Filed: **Dec. 23, 1994**

Related U.S. Application Data

[63] Continuation of Ser. No. 832,944, Feb. 10, 1992, abandoned.

[51] Int. Cl.⁶ **G06F 12/10**

[52] U.S. Cl. **395/418; 395/421.02**

[58] Field of Search 395/400, 416,
 395/417, 418, 421.02; 364/DIG. 1, 254.9,
 255.7, 256.4

[56] References Cited

U.S. PATENT DOCUMENTS

| | | | |
|-----------|---------|-----------------|---------|
| 4,340,932 | 7/1982 | Bakula et al. | 395/402 |
| 4,432,053 | 2/1984 | Gaither et al. | 395/416 |
| 4,654,777 | 3/1987 | Nakamura | 395/416 |
| 4,669,043 | 5/1987 | Kaplinsky | 395/403 |
| 4,679,140 | 7/1987 | Gotou et al. | 395/775 |
| 4,758,946 | 7/1988 | Shar et al. | 395/416 |
| 4,763,250 | 8/1988 | Keshlear et al. | 395/418 |
| 4,792,897 | 12/1988 | Gotou et al. | 395/417 |
| 4,835,734 | 5/1989 | Kodaira et al. | 395/419 |
| 4,972,338 | 11/1990 | Crawford et al. | 395/416 |
| 4,979,098 | 12/1990 | Baum et al. | 395/418 |
| 5,023,777 | 6/1991 | Sawamoto | 395/402 |
| 5,263,140 | 11/1993 | Riordan | 395/417 |
| 5,475,827 | 12/1995 | Lee et al. | 395/417 |

FOREIGN PATENT DOCUMENTS

| | | |
|---------|---------|--------------------|
| 0113240 | 12/1983 | European Pat. Off. |
| 1595740 | 5/1978 | United Kingdom |
| 2127994 | 6/1983 | United Kingdom |

OTHER PUBLICATIONS

Data General MV 2000, Chapter 3, *Logical To Physical Address Translation*; pp. 32-37 (publication information unknown).

i860™ Microprocessor Family Programmer's Reference Manual, Intel Corporation Literature Sales, P.O. Box 7641, Mt. Prospect, Ill 650056-7641, Chapter 4, pp. 1-13, 1991.
 Patterson, David A. and John L. Hennessey, *Computer Architecture A Quantative Approach*, Morgan Kaufman Publishers, Inc., San Mateo, California pp. 432-485.

(List continued on next page.)

Primary Examiner—Eddie P. Chan

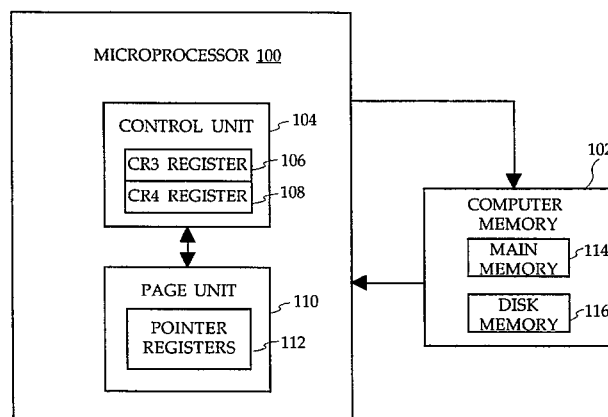
Assistant Examiner—Reginald G. Bragdon

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] ABSTRACT

An address translator and a method for translating a linear address into a physical address for memory management in a computer is described herein. Different memory sizes, and different page sizes can be selected. The address translator can translate from a standard 32-bit linear address for compatibility with previous 32-bit architectures, and can also translate to a physical memory size with a larger physical address than linear address; i.e., greater than 32 bits (e.g. 36 bits and up), with no increase in access time. The address translator translates a linear address that includes an offset and a plurality of fields used to select entries in a plurality of tables. The format of the linear address into fields is dependent upon the selected memory size and the selected page size. For a large memory size, the tables include a directory pointer table that includes a group of directory pointers, a plurality of page table directories each of which includes a group of page directory entries, and a plurality of page tables each of which includes a group of page table entries. The size of the entries in the tables is dependent upon the selected memory size. The contents of the tables are stored in memory, and furthermore the pointer table is stored in both main memory and in dedicated pointer table registers.

28 Claims, 12 Drawing Sheets



5,617,554

Page 2

OTHER PUBLICATIONS

Nelson, Ross P., *The 80386 Book*; Chapter 6, Memory Architecture; Paging, Microsoft Press, pp. 125-134, 1988.
i486™ Processor Programmer's Reference Manual, Chapter 5, Intel Corporation Literature Sales, P.O. Box 7641, Mt. Prospect, Ill 60056-7641, pp. 1-25, 1990.

i860™ XP Microprocessor Data Book, Chapter 2, Intel Corporation Literature Sales, P.O. Box 7641, Mt. Prospect, Ill 60056-7641, pp. 21-27, 1991.

"Sun Microsystems: The SPARC™ Architecture Manual Version 8"; Sun Microsystems, Inc., 11 Dec. 1990.

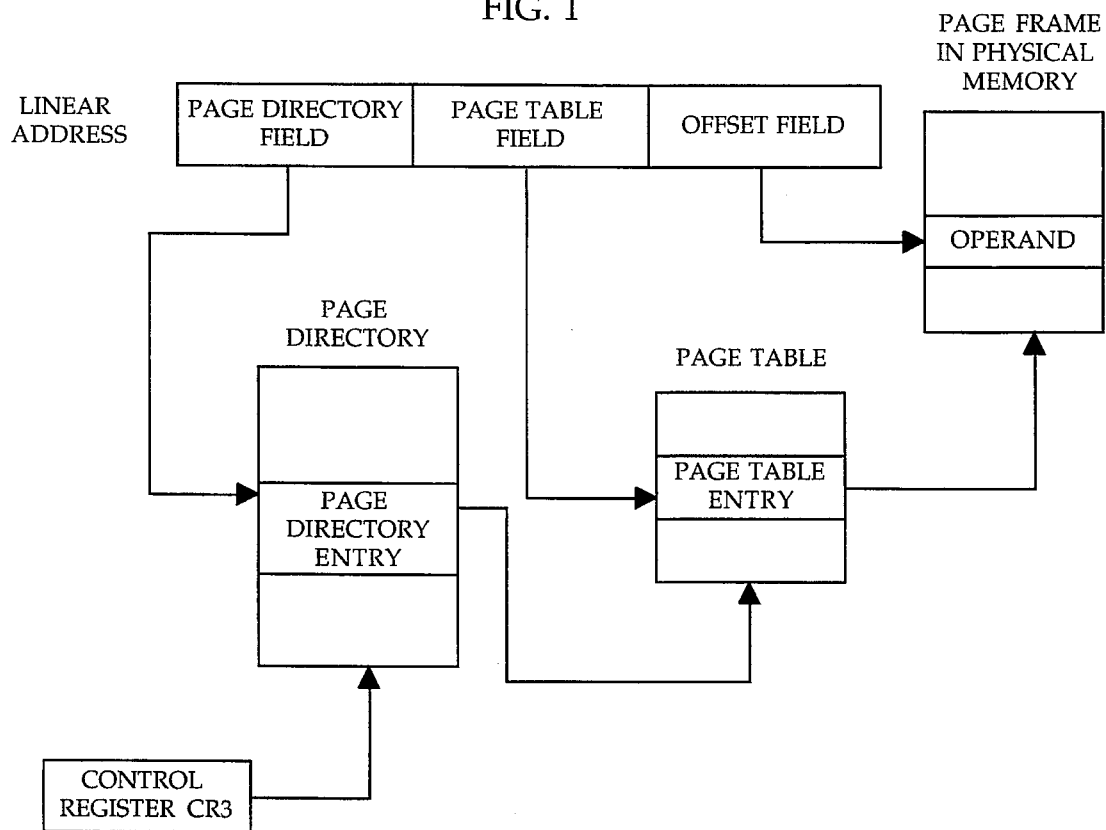
U.S. Patent

Apr. 1, 1997

Sheet 1 of 12

5,617,554

FIG. 1

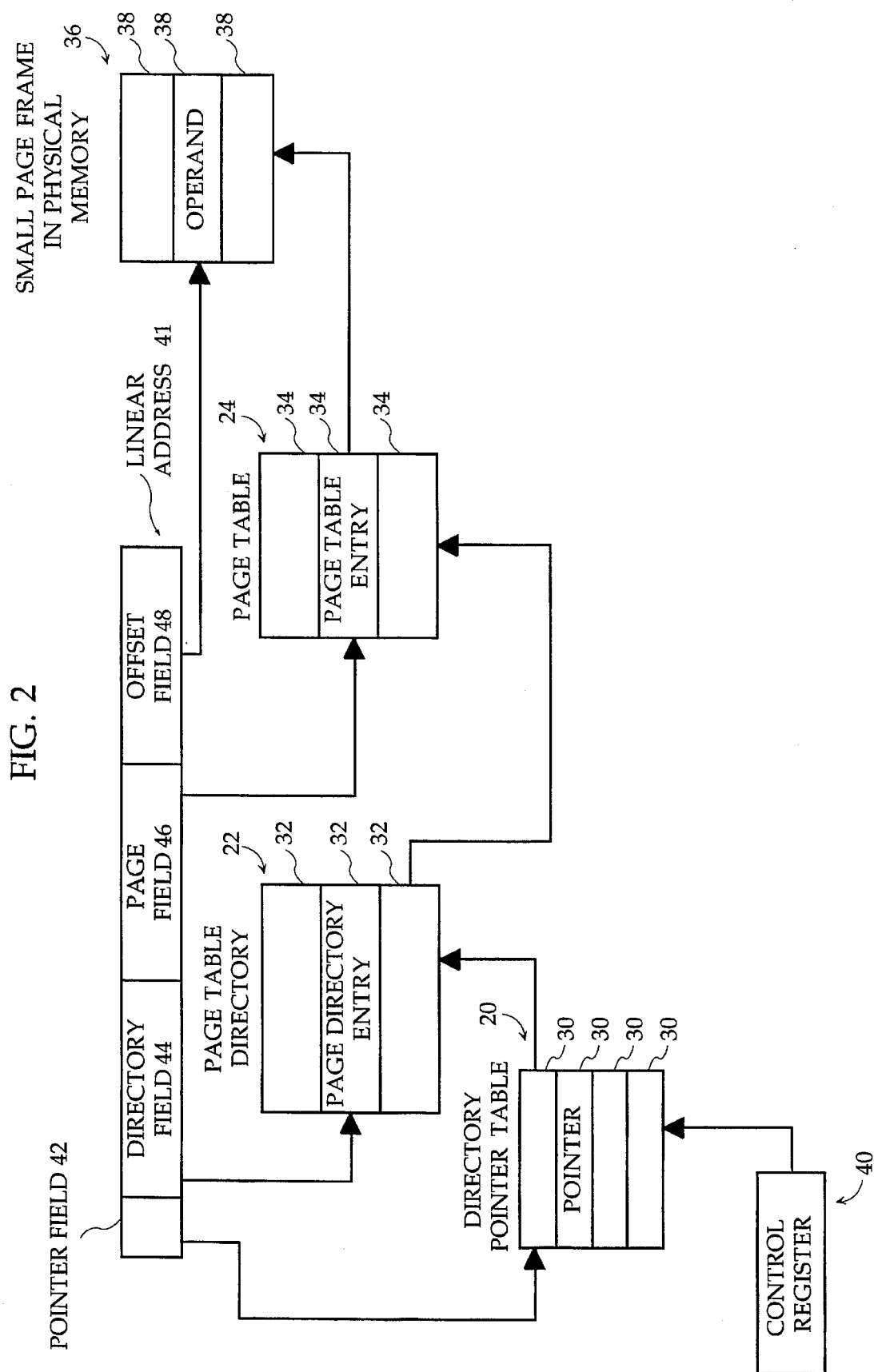


U.S. Patent

Apr. 1, 1997

Sheet 2 of 12

5,617,554



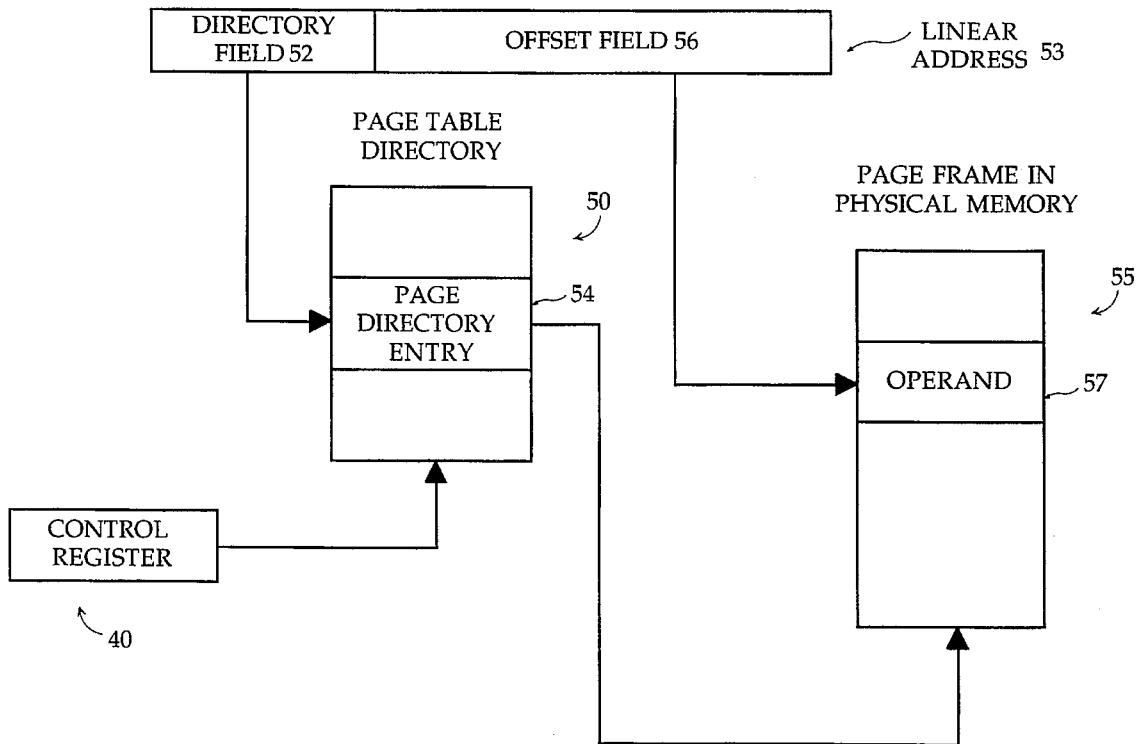
U.S. Patent

Apr. 1, 1997

Sheet 3 of 12

5,617,554

FIG. 3



U.S. Patent

Apr. 1, 1997

Sheet 4 of 12

5,617,554

FIG. 4

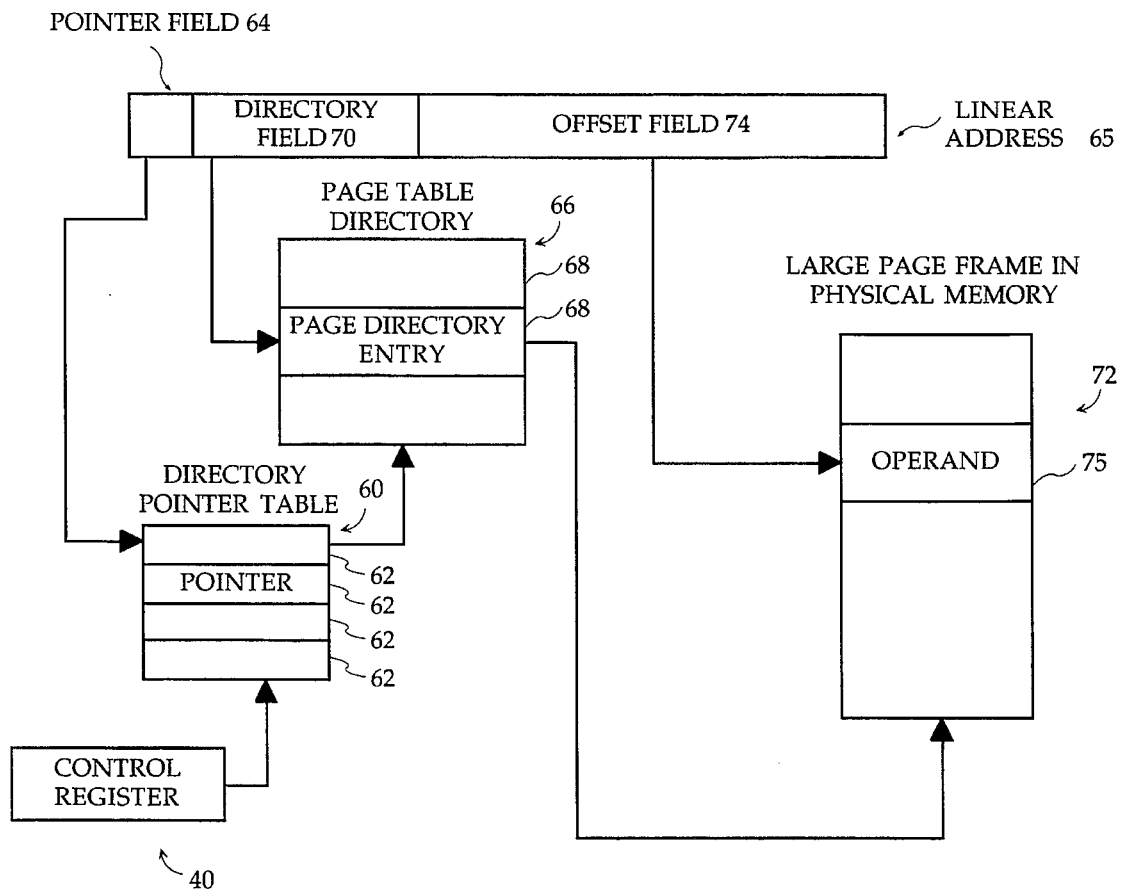


FIG. 5

 $\angle 20^\circ$

| PAGE DIRECTORY ADDRESS | | | | | | | | | | | | P C D | P W T | | | |
|------------------------|---|---|---|---|---|---|---|---|---|---|---|-------------|-------------|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | 0 | 0 | 0 |

CONTROL REGISTER 3 FORMAT

 $\angle 20^\circ$

Λ
3
V

[illegible]

PAGEDIRECTORYENTRYFORMAT

 $\angle 20^\circ$

Λ
3
V

[illegible]

PAGE TABLE ENTRY FORMAT

FIG. 6

< 20 >

| | | | | | | | | | | | | |
|------------------------|---|---|---|---|---|---|---|---|---|---|---|---|
| PAGE DIRECTORY ADDRESS | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | P | P | P | 0 |
| | | | | | | | | | C | W | 0 | 0 |
| | | | | | | | | | D | T | | |

CONTROL REGISTER 3 FORMAT

< 20 >

< 3 >

| | | | | | | | | | | | | | | | | | | | |
|--------------------|--|--|--|--|--|--|--|--|--|------------|-------------|--------|---|---|-------------|-------------|---|---|---|
| PAGE TABLE ADDRESS | | | | | | | | | | OS RES. | R E S | P S | 0 | A | P C D | P W T | U | W | P |
|--------------------|--|--|--|--|--|--|--|--|--|------------|-------------|--------|---|---|-------------|-------------|---|---|---|

PAGE DIRECTORY ENTRY FORMAT (PDE.PS=0)

< 20 >

< 3 >

| | | | | | | | | | | | | | | | | | | |
|--------------------|--|--|--|--|--|--|--|--|--|------------|------------------------------|---|---|-------------|-------------|---|---|---|
| PAGE FRAME ADDRESS | | | | | | | | | | OS RES. | ² RES. BITS | D | A | P C D | P W T | U | W | P |
|--------------------|--|--|--|--|--|--|--|--|--|------------|------------------------------|---|---|-------------|-------------|---|---|---|

PAGE TABLE ENTRY FORMAT

FIG. 7

 $\angle > 20^\circ$

| PAGE DIRECTORY ADDRESS | | | | | | | | | |
|------------------------|---|---|---|---|---|---|---|-------------|-------------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | P C D | P W T |

CONTROL REGISTER 3 FORMAT

 $\sqrt{10}^{\wedge}$

Λ
3
V

| | | | | | | | | | | |
|-----------------------|------------------|---------|-------|-----|---|---|-------|-------|---|---|
| 4M PAGE FRAME ADDRESS | 10 RESERVED BITS | OS RES. | R E S | P S | D | A | P C D | P W T | U | P |
|-----------------------|------------------|---------|-------|-----|---|---|-------|-------|---|---|

PAGEDIRECTORYENTRYFORMAT(PDE.PS=1)

FIG. 8

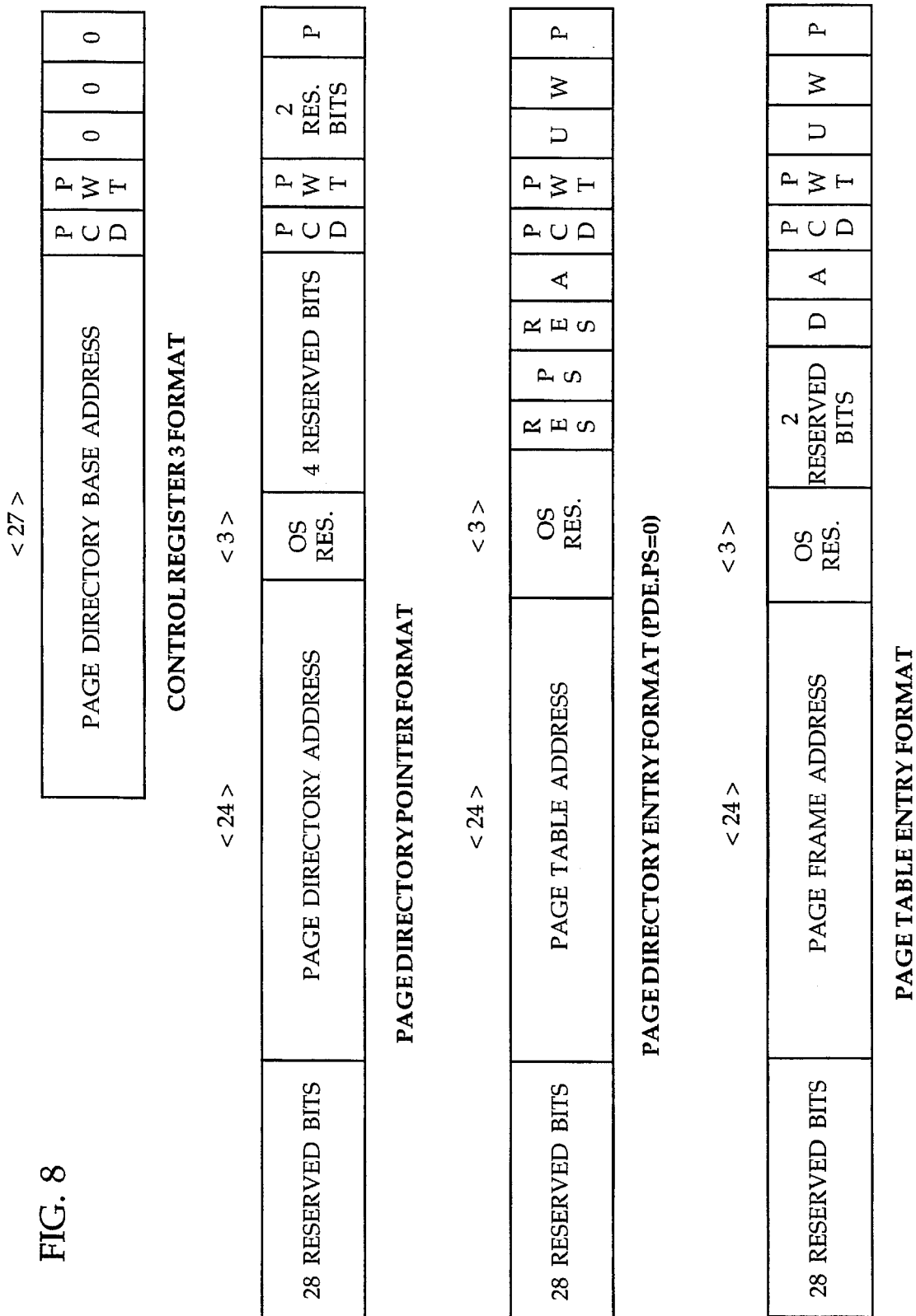
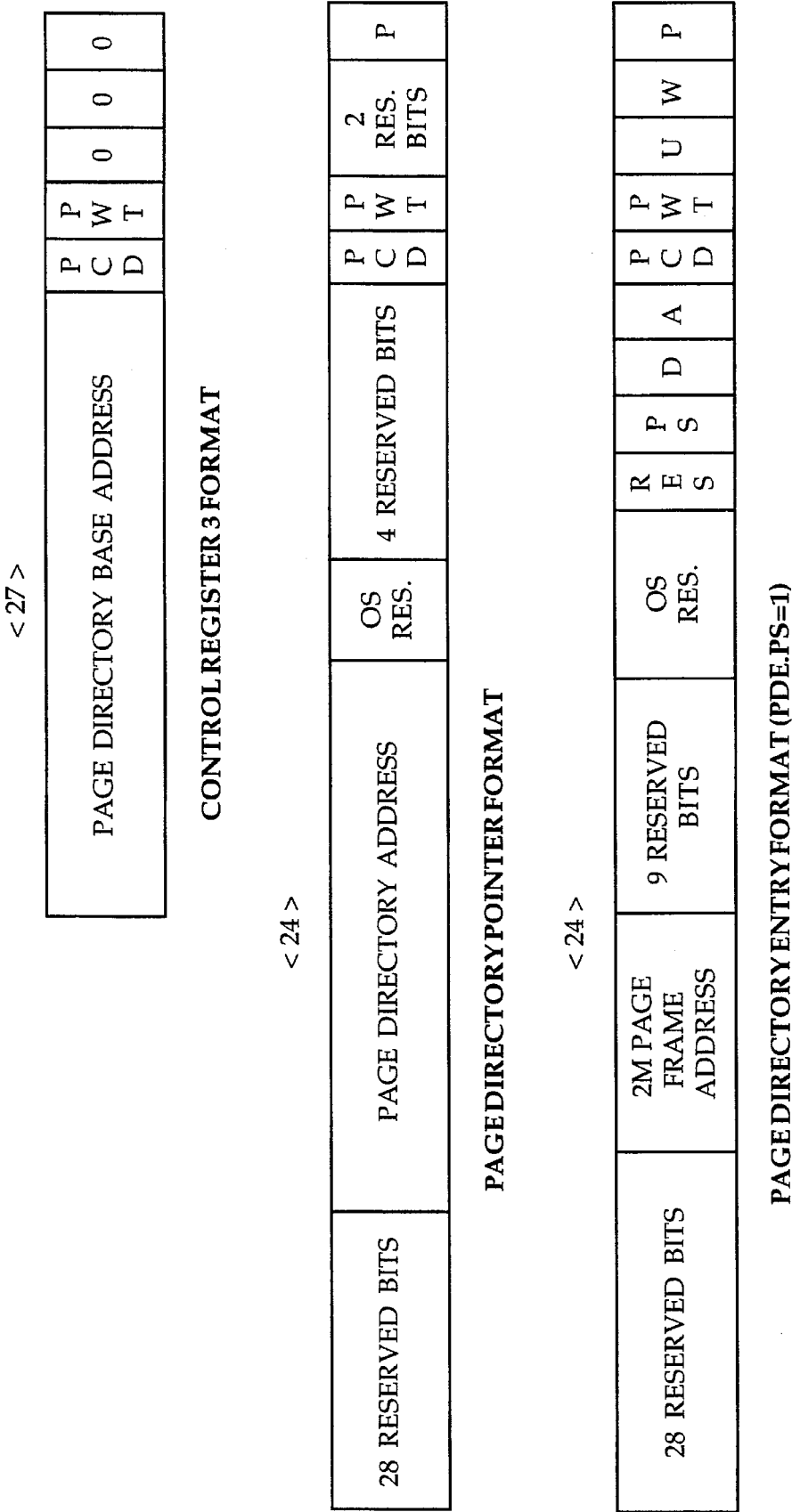


FIG. 9



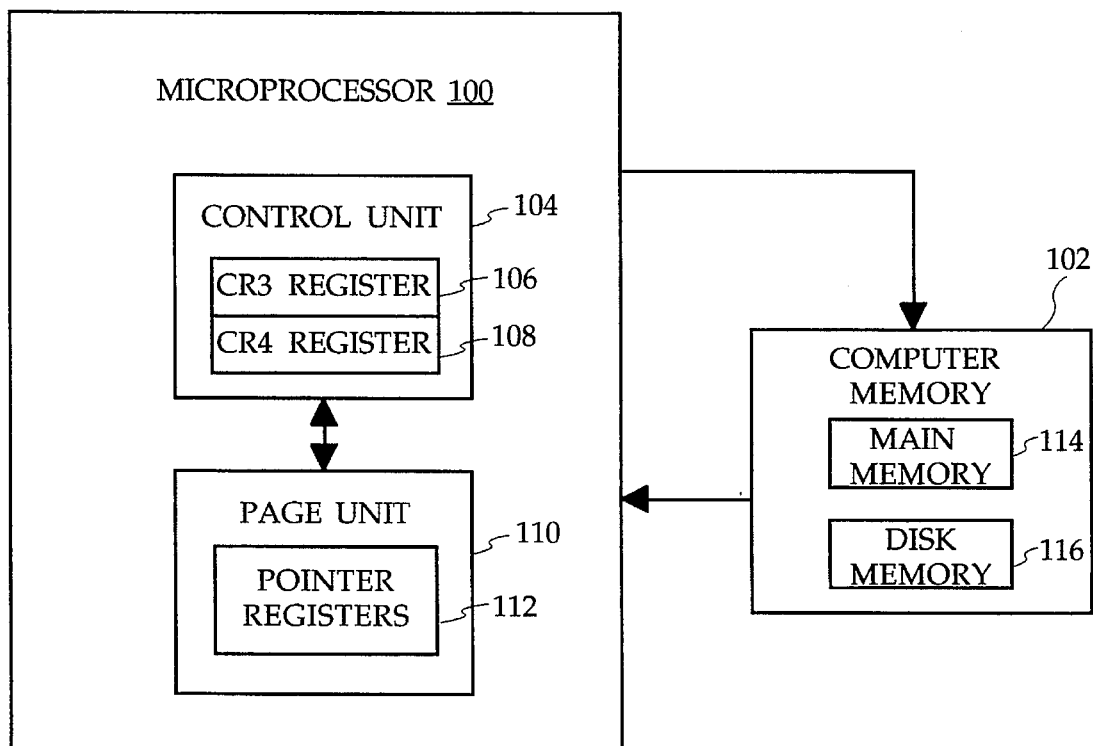
U.S. Patent

Apr. 1, 1997

Sheet 10 of 12

5,617,554

FIG. 10



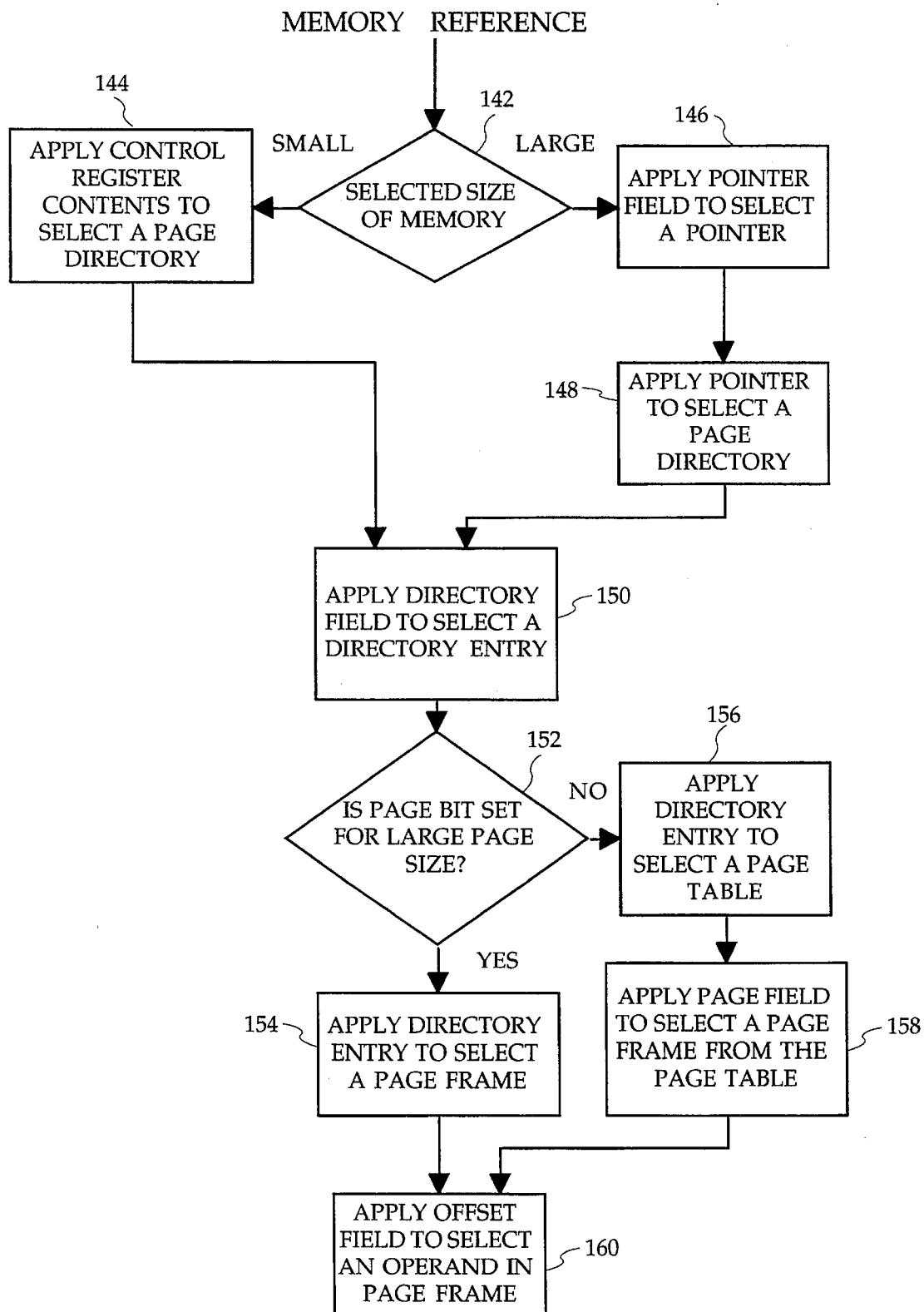
U.S. Patent

Apr. 1, 1997

Sheet 12 of 12

5,617,554

FIG. 12



5,617,554

1

PHYSICAL ADDRESS SIZE SELECTION AND PAGE SIZE SELECTION IN AN ADDRESS TRANSLATOR

This is a continuation of application Ser. No. 07/832,944, 5
filed Feb. 10, 1992, now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to computer memory management methods and apparatus for translating between a virtual address and a physical address stored in a plurality of pages in main memory.

2. Description of Related Art

A computer typically includes main memory and secondary memory. A microprocessor may be included to control the writing and reading of these memory elements. Main memory, constructed of RAM (Random Access Memory) chips is much faster than secondary memory such as a hard disk drive, but main memory is typically much more expensive per memory element. Thus, computers are usually designed with a main memory limited in size and a much larger secondary memory. Microprocessors designed to use virtual memory effectively extend main memory space into secondary memory space. Typically, virtual memory microprocessors use techniques such as paging or segmentation, or both, to simulate a larger memory.

Virtual memory provides many advantages. It is possible to run applications that require more RAM (Random Access Memory) than is actually available. Furthermore, it is possible for more than one application to run at the same time. Virtual memory gives an applications programmer a view of more main memory than actually exists. To a programmer, virtual memory appears as one contiguous block of main memory. An applications programmer seeking storage space does not have to concern himself with the actual physical location of that data, whether the data is in main memory, or on a hard disk.

In virtual memory systems, the term "physical memory" is used to define the physical address space seen by the operating system. Thus, the size of the physical address space will often be larger than the main memory available in the system. However, the data at any specific physical address must be accessed within the main memory. In operation of virtual memory systems, if requested data is not in main memory, then an operating system exception will be generated and the data will be transferred from secondary storage to main memory under the auspices of the operating system. If paging has been implemented, then a page validity bit may be associated with each page to indicate whether or not the page is in main memory. If the page validity bit indicates that a requested page is not in main memory, then the data is transferred to main memory and the page validity bit is set to indicate that the page is now in main memory.

Virtual memory systems can be categorized into two types: those with variable sized blocks, called "segments", and those with fixed size blocks, called "pages". One, or a combination of these two different methods are used to translate from a virtual address to a physical address. Using segmentation, a block of physical memory is allocated based on the amount of data to be stored. Thus, in a segmented memory structure, one segment may be large and the next may be very small. Segmentation uses memory efficiently, however there are disadvantages. Problems arise when a segmented data structure is modified to be larger. The

2

smaller block must be replaced, and a new, larger block must be found. This problem is substantial, particularly when it is recognized that data structures are often modified.

Paging is the other method for translating a virtual address to a physical address. Pages are fixed size blocks of memory that are mapped in specific locations in physical memory. A "page" is what the programmer sees (part of the logical or linear address), and the "page frame" is the physical memory itself. One or more tables are provided, each of which has a number of page table entries. Each page table entry specifies a specific page frame. The virtual address includes information that is indicative of the table and the page frame.

The use of pages provides advantages. Paging is generally efficient; there are no unused blocks but internal fragmentation (i.e., an unused portion of a page) can be a problem. Replacing a block is trivial. If data is modified to include additional information, then additional pages can be employed. From a design point of view, paging is generally preferred for bigger systems because paging makes allocation of memory easier.

The page size is an important architectural parameter. Choosing a page size is a question of balancing forces that favor a larger page size versus those favoring a smaller size. Advantages of larger page size include memory resources that are saved by use of larger page size, and efficient transfer of larger pages to and from secondary storage. Particularly, memory resources are saved because the size of the page table is inversely proportional to the page size, and thus a larger page size means a smaller table. Also, transferring larger pages is more efficient than transferring smaller pages. For example, the page size for the Intel 80386 and the i486™ microprocessor is 4 Kbytes.

Most microprocessors employ a combination of segmentation and paging; specifically, each segment includes a number of pages. This approach to memory management is termed "paged segmentation". Such a system is disclosed in U.S. Pat. No. 4,972,338, entitled "Memory Management For Microprocessor System" issued to Crawford et al., which discloses a segmentation mechanism for translating a virtual memory address to a second memory address (linear address) that is applied to a two-level paging table to select a page frame. The Crawford et al. invention is embodied in the Intel 80386 microprocessor.

Briefly, as described in the Crawford et al. patent, segmentation translates a 48-bit virtual address to a 32-bit linear (intermediate) address. The virtual address includes a segment selector (14 bits) and segment offset (32 bits). Segment offset is the calculated result of the address calculation: scale index and the displacement. The segment selector comes from the segment register. The application programmer must specify a segment register and other components that give the offset. Most applications use only one segment.

FIG. 1 is an illustration of page translation in Intel 80386 microprocessor and the Intel i486™ microprocessor. A linear address including a page directory field, a page table field, and an offset field are applied to obtain an operand in a page frame. Within the computer memory, the data structures include page directories, page tables, and page frames. In operation, a page directory in memory is first chosen by a control register CR3. In the linear address, the page directory field selects a page directory entry that specifies a page table in physical memory. The page table field of the linear address selects a page table entry that specifies a page frame of data in physical memory. The offset field of the linear address is applied to select an operand within the page frame. Thus, page translation for the Intel i486™ uses a tree structure with two levels of tables in memory.

5,617,554

3

Each of the tables occupies a 4 Kbyte block of physical memory and each entry is 4 bytes. Therefore each page directory address table has 1K 4-byte entries. The directory field of the linear address is a 10-bit index that selects one of the 1K entries in memory. Similarly, the page table has 1K 4-byte entries that are selected by the 10-bit page table field of the linear address. The 12-bit offset field of the linear address is applied directly to select any operand in the 4K bytes in the page frame. The page frame address is 20 bits in memory. With the 12 bit offset of the linear address, the translation provides a 32-bit physical address, which can access up to 4 Gbytes of physical memory.

Although large by past standards, a 4 Gbyte limit on physical memory is becoming a limitation, particularly for very large servers. For example, banking databases that store customer information may require tens of Gbytes. It would be an advantage to provide a memory management system that is compatible with 32-bit existing software for the 386 and i486™, and can access more than 4 Gbytes, for example up to 64 bits (1.8×10^{10} Gbytes). It would be a performance advantage if the memory access to this larger physical memory could be completed in the same, or a lesser amount of time. It would be another advantage if more than one page size could be selected to choose a page size corresponding to an efficient memory structure for a particular application program or type of data structure.

SUMMARY OF THE INVENTION

The present invention provides an address translator and a method for translating a linear address into a physical address for memory management in a computer. Different page sizes can be selected, and the address translator translates the selected page size. The address translator translates from a standard 32-bit linear address for compatibility with previous architectures. However, the translator can translate to a physical address that is larger than the linear address; i.e., greater than 32 bits (e.g., 36 bits). Translation occurs with no decrease in performance, i.e., no increase in access time. In alternative embodiments, the address translator can be expanded to accommodate greatly increased physical memory while retaining the ability to run pre-existing software. For example, the physical address of the preferred embodiment can be expanded to meet a new IEEE standard which requires a 64-bit physical address. Even with such a large physical address, compatibility is retained with systems that use a 32-bit physical address.

The apparatus of the present invention includes a memory management address translator for addressing a page frame of main memory using another computer supplied address. In the preferred embodiment, the translator translates a linear address supplied by a microprocessor in a computer that uses segmentation methods to obtain the linear address by translating a virtual address. However, it should be apparent to one skilled in the art that, in other computer architectures, the translator of the present invention could translate a computer supplied address obtained by another method.

The linear address that is applied to the translator includes an offset and a plurality of fields used to select entries in a plurality of tables. The tables include a directory pointer table that includes a group of directory pointers, a plurality of page table directories each of which includes a group of page directory entries, and a plurality of page tables each of which includes a group of page table entries. The size of the entries in the page table directory is dependent upon the

4

selected memory size, and is either 32-bits or 64-bits in the preferred embodiment. A first field of the linear address specifies a directory pointer in the directory pointer table, so that a directory pointer is selected that specifies a page directory table. A second field of the linear address specifies a page directory entry in the selected page directory table. A third field of the linear address specifies a page table entry so that a page frame in physical memory is selected.

The contents of the tables are stored in any suitable memory component such as main memory. The page table directories and the page tables are stored in main memory and accessed through a conventional memory access. In the preferred embodiment, the pointer table is stored in both main memory and in dedicated pointer table registers provided in the microprocessor. Preferably, the pointer table registers are invisible (i.e. not accessible) to a computer programmer, so that the registers cannot be written to or read directly by the programmer. A performance advantage is provided by the use of registers for the directory pointer table because a memory access is avoided.

In comparison to the memory management system of the i486™ microprocessor, the address translator described herein includes the directory pointer table in addition to the page table directories and the page tables. As a further feature, the directory pointer table is in main memory and also in dedicated on-chip registers. Register storage avoids one memory access for each translation, so that accessing the three tables to translate an address requires only two memory accesses at most, thereby saving computer time. Only one page size was supported on the i486™ microprocessor, but in the present invention, more than one page size is available. The size of the linear address remains 32 bits, however the table structure allows the physical address to be larger. For example, with extended addressing the physical address may be 36 bits (64 GB) and can be larger, 64 bits for example. The same size linear address permits present operating systems to take immediate advantage of extended physical addressing without change to their system software.

A larger page size can provide performance advantages in some instances, particularly for large amounts of contiguous data, by reducing the miss rate of the Translation Lookaside Buffer (TLB). With a larger page size, a single TLB entry can be used to access all the locations in the large page instead of using separate entries for accessing each small page. The single TLB entry for a large page saves time by avoiding the multiple TLB accesses that would occur if a small page size were used. Larger page sizes are useful, for example, in mapping memory-resident portions of the operating system, video display frame buffers, databases, and other large data structures. A large page size allows efficient reading and storing of these data structures.

In memory, a page is defined by a page frame that has boundaries set by its size. For example, a 4 Kbyte page has boundaries at each 4 Kbyte memory boundary, and a 2 Mbyte page has boundaries at each 2 Mbyte memory boundary. As discussed above, the address translator supports multiple page sizes; i.e. multiple page frame sizes. A single program may use multiple page sizes. The supported page frame sizes include: a first page frame size and a larger, second page frame size. The linear address includes a plurality of fields including selection fields and an offset field. The number and width of the selection fields is dependent upon the selected page frame size. For the first page frame size, the selection fields include a first selection field, a second selection field, and a third selection field. For a second page frame size, the selection fields include only a

5,617,554

5

first selection field. For a second page frame size with extended physical addressing, a second selection field is used in addition to the first selection field.

If the first page size is selected, then all three tables are used to translate a linear address to a physical address. However, if the second, larger page size is selected, then at least one of the tables is omitted. In the preferred embodiment with the normal addressing mode, the page tables and the pointer table are not used, and instead, the page directory entry points directly to the larger page frame. With extended physical addressing, the pointer tables and the page directory tables are used, but the page tables are not used and the page directory entry has a larger size and points directly to the larger page frame.

When the first page size is selected so that the linear address includes the first, the second, and the third selection field, the first selection field selects a directory pointer entry to select a directory table, the second selection field selects a selected page directory entry from the selected page directory table to select a page table, and the third selection field selects a page frame address from the selected page table. When the second page size and normal addressing is selected so that the linear address comprises only a first selection field and an offset field, the control register selects a page directory, and the first selection field points to a page frame. When the second page size and extended physical addressing is selected so that the linear address comprises a first field and a second field, the first selection field of the linear address selects a directory pointer to select a page directory table, and the second selection field selects a page frame from the selected page directory table.

In terms of a method, the present invention includes an address translation method for translating a linear address into a physical address. The translation method varies dependent upon page size and size of the physical address. Thus, two selections are made initially. A first selection selects either a small or a large page size. A second selection selects either normal addressing or extended physical addressing. The format of the linear address is selected to correspond to the selected page size although preferably the linear address size remains unchanged.

For the small page size in the described embodiment, the linear address has a pointer field, a directory field, a page field, and an offset field. The translation method for the small page size comprises the steps of applying the pointer field to select a directory pointer, applying the selected directory pointer to select a page directory, applying the page directory field to the selected page directory table to select a page table, applying the page table field to the selected page table to select a page frame address, and applying the page frame address and the offset field to provide a physical address.

If the second page size and normal addressing has been selected, then the second page size translation includes applying a control register to select a page directory, applying the first selection field in the linear address to the page directory to select a page frame, and applying the offset to select a particular operand. If the second page size and extended physical addressing has been selected, then the second page size translation method includes the steps of applying a directory pointer field to select a directory pointer, using the selected directory pointer to select a page directory, applying the page directory field to said selected page directory table to select a page frame address, and using the page frame address and the offset field to provide a physical address.

6

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagrammatic illustration of page translation for the i486™ microprocessor.

FIG. 2 is a diagrammatic illustration of page translation for the translator of the present invention when a small page frame size and extended addressing has been selected.

FIG. 3 is a diagrammatic illustration of page translation for the translator of the present invention when a large page frame size and normal addressing has been selected.

FIG. 4 is a diagrammatic illustration of page translation for the translator of the present invention when a large page frame size and extended physical addressing has been selected.

FIG. 5 is a diagram of a control register, a page directory entry, and a page table entry for a normal physical address space and a single page size that is the original and default mode for the Intel 80×86 family.

FIG. 6 is a diagram of a control register, a page directory entry, and a page table entry for a normal physical address space and a small page size.

FIG. 7 is a diagram of a control register and a page directory entry for a normal physical address space and a large page size.

FIG. 8 is a diagram of a control register, a page directory pointer, a page directory entry, and a page table entry for an extended physical address space and a small page size.

FIG. 9 is a diagram of a control register, a page directory pointer, and a page directory entry for an extended physical address space and a large page size.

FIG. 10 is a block diagram of a portion of the microprocessor and its connection to memory elements.

FIG. 11 and 12 together show a flow chart of address translation for the microprocessor of the preferred embodiment, including selectable page size and memory size.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT OF THE INVENTION

The invention may be understood by reference to the following detailed description of the preferred embodiment and the figures wherein like parts are designated with like numerals throughout.

The preferred embodiment of the present invention includes a plurality of page tables at different levels whose structure and addressing is dependent upon the chosen page size and the number of physical address bits that are desired for the physical address. In the preferred embodiment, the tables are stored in digital form in memory, except where otherwise noted. The circuits necessary to perform the processing to be described can be implemented using a finite state machine, for example a programmable logic array (PLA), and other conventional logic.

Table 1, shown below, discloses features of the preferred embodiment. Specifically, Table 1 discloses selection of the page size and the size of the physical address. In the preferred embodiment, selection is controlled by two control bits in a control register (CR4). The two control bits are termed the "PAE" bit and the "PSE" bit.

5,617,554

7

TABLE 1

| Memory Management Features Controlled by PSE and PAE | | | |
|--|-----|--------------------|-----------------------|
| Control Bits | | Features Available | |
| PAE | PSE | Page Sizes | Physical Address Bits |
| 0 | 0 | 4 KB | 32 |
| 0 | 1 | 4 KB or 4 MB | 32 |
| 1 | 0 | 4 KB or 2 MB | 36 |
| 1 | 1 | 4 KB or 2 MB | 36 |

The address mode is selected by the PAE bit. If PAE is "0", then normal 32-bit addressing is selected. If PAE is "1", then extended physical addressing is selected.

If PAE and PSE are both zero, then the page size is 4 KB and the physical address is 32-bits. This configuration is compatible with memory management in microprocessors including the Intel 80386 microprocessor and the 80486 microprocessor, both of which are commercially available and in wide use. Thus, software written for those microprocessors can be nm with the memory management of the preferred embodiment, with PAE and PSE both set to zero.

If PAE is zero and PSE is set to one, then the size of the physical address is again 32-bits. However, the page size may be either 4 KB or 4 MB, depending upon a control bit in the page directory entry to be described below. The larger page size may be useful in applications where large blocks of memory are stored together, for example, video memory. In such an instance, the program sets the control bit when the page directory entry is entered into the directory.

If PAE is one and PSE is either set to one or reset to zero (i.e., "don't care"), then the physical address is extended to 36-bits, and the page size may be either 4 KB or 2 MB, depending upon a control bit in the page directory entry to be described. The 36-bit address provides extended addressing capabilities (64 Gbytes in the preferred embodiment) to address more memory than commercially available microprocessors including the Intel 80386 microprocessor and the 80486 microprocessor.

The above description and Table 1 show specific parameters applicable to the preferred embodiment of the invention. In other embodiments, the page size may be different, the number of control bits may be more or less, and the size of the physical address may be different. The smaller physical address size can be chosen to be compatible with prior art microprocessors, and the larger physical address size can be chosen for use with larger memory. In the preferred embodiment, the size of the physical address can be expanded by simply utilizing bits that are currently not utilized in the embodiment described herein. The physical address of 32-bits can address each memory block in a 4 GByte physical memory, while the extended physical address of 36-bits can address each memory block in a 64 GByte physical memory.

In the preferred embodiment, the physical address is translated from a linear address supplied by a microprocessor. In the preferred embodiment, the linear address is supplied by applying segmentation to a virtual address, however it will be apparent to one skilled in the art that the linear address may be supplied using other methods. The linear address is divided into fields used to translate the linear address into a physical address. Both the number of fields in the linear address, and the size of the fields may vary dependent upon the selected page size and physical address size, as will be described. In the preferred embodi-

8

ment, the number of fields and their size are both dependent upon the control parameters shown in Table 1, specifically, the PAE and PSE control bits, which specify the page size and the physical address size.

For page translation using a small 4K page size and a normal physical address space (4 GByte or less), the translation configuration is similar to that shown in FIG. 1, used in the Intel 80386 and 80486. That configuration is well known and described in many books, for example "Programming the 80386", by John H. Crawford and Patrick P. Gelsinger, Sybex, San Francisco, 1987.

Reference is made to FIG. 2 which is an illustration of page translation for a small page size and an extended physical memory. In the preferred embodiment, the small page size is 4 KB and the extended physical memory is 64 GByte or less. For the small page size and extended physical memory, three levels of tables are used. These three levels include a directory pointer table 20, a page table directory 22 and a page table 24. The directory pointer table 20 includes a plurality of pointer entries 30, the page table directory 22 includes a plurality of page directory entries 32, and the page table 24 includes a plurality of page table entries 34. A page frame 36 represents a block of data in memory having a size determined by the page directory entry 32, and by the PAE and PSE bits as described above with reference to Table 1. Each page frame 36 includes a plurality of operands 38 that are selected in a manner described below.

In order to access an operand, a control register 40 holds a value that points to the address of a particular directory pointer table 20 in memory. In the preferred embodiment, the directory pointer table 20 is actually stored in the microprocessor in dedicated registers, as will be described below. A linear address 41 for a small page size is divided into a pointer field 42, a directory field 44, a page field 46, and an offset field 48.

The pointer field 42 points to a particular pointer 30 in the directory pointer table 20 that is selected by the control register 40. In the preferred embodiment, the directory pointer table 20 includes four entries, and the pointer field 42 therefore includes 2-bits. The selected pointer 30 is applied to select a particular page table directory 22. Once the page table directory 22 is selected, a directory field 44 of the linear address selects a particular page directory entry 32. The selected page directory entry 32 points to one of the plurality of page tables 24. A page field 46 within the linear address selects a particular page table entry 34 in the selected page table 24. The selected page table entry 34 selects a page frame 36 in physical memory. The offset field 48 in the linear address points to an operand 38 in the specified page frame 36 in physical memory.

Reference is made to FIG. 3 which is an illustration of page translation for a larger (4 MB) page size with a normal size (4 GByte or less) physical memory. For translating the large page size, only one table level is used in place of the three levels of tables for the smaller size page as discussed previously with regard to FIG. 2. The control register 40 points directly to a page table directory 50. A directory field 52 in a linear address 53 points to a particular entry in the page directory. The format of the page directory entry will be described below. The selected page directory entry 54 points to a page frame 55 in physical memory. An offset field 56 in the linear address points to a particular operand 57 within the page frame 55.

Reference is made to FIG. 4 which is a configuration for pointing to a large page frame in physical memory. The embodiment of FIG. 4 is useful in the preferred embodiment

5,617,554

9

for extended addressing (64 GBytes or less). In other words, when a larger physical memory is provided, the embodiment of FIG. 4 provides additional information necessary to access the data within the larger physical memory. In the configuration of FIG. 4, two levels of tables are used, with the first level of tables being a directory pointer table 60. The control register 40 selects a directory pointer table 60 that includes a plurality of pointers 62. The format of the pointer will be described below. A pointer field 64 within a linear address 65 points to a particular pointer 62. The selected pointer points to a page table directory 66 resident within memory. A plurality of page directory entries 68 are included within the page table directory 66. The format of the page directory entries 68 will be described below. A directory field 70 within the linear address points to a particular page directory entry 68, which in turn selects a particular page frame 72 in physical memory. An offset field 74 in the linear address selects a particular operand 75 within the page frame 72. In the preferred embodiment, the size of the page frame 72 for extended addressing is 2 MB.

10

the page directory and will come from bits 31:12 of CR3. The offset within the page directory will come from the ten most significant bits of the linear address (31:22). One 4K-byte page can therefore fit 1K of 4-byte entries.

The details of operation in the default mode are well known with respect to Intel's 80386 and 80486 microprocessors and will not be repeated here. Reference is made to FIG. 1, which shows the table configuration that is implemented with the control register and table entries shown in FIG. 5.

Reference is now made to FIG. 6 which is a depiction of the format of the control register and the table entries. The look-up sequence for type 1 is virtually identical to that described for type 0 with two minor exceptions: the reserved (RSV) bits are checked for zero as soon as the P bit is validated, and the bit PDE.PS (Page Directory Entry Page Size) is zero for a small page size in type 1. (For a large page size, PDE.PS=1 which is type 2.) The table configuration for type 0 shown in Fig. 1 is also the table configuration for type 1.

TABLE 2

Memory Management Types and Modes Run

| Type | CR4 PAE | CR4 PSE | DIR PS | RELATED FIG. | PAGE SIZE (Bytes) | PHYSICAL ADDRESS | MODE |
|------|------------|------------|-----------|-----------------|-------------------------|---------------------|---------|
| 0 | 0 | 0 | X | FIG. 5 | 4K | Normal (32-bit) | A (Def) |
| 1 | 0 | 1 | 0 | FIG. 6 | 4K | Normal (32-bit) | B |
| 2 | 0 | 1 | 1 | FIG. 7 | 4M | Normal (32-bit) | B |
| 3 | 1 | X | 0 | FIG. 8 | 4K | Extended (36-bit) | C |
| 4 | 1 | X | 1 | FIG. 9 | 2M | Extended (36-bit) | C |

For a more detailed description of the preferred embodiment, Table 2 sets forth memory management types and the modes that are run. Mode A is the default mode. This is the mode that is compatible with all other members of the X86 family of microprocessors, including the Intel 80386 and the Intel 80486 microprocessors. Mode B represents normal addressing (<4 GByte) and a capability for selecting one or two page sizes. Mode C represents extended addressing (a larger physical memory) and a capability for selecting one or two page sizes. Memory management for these modes has been discussed with reference to FIGS. 1-4. Each of these modes will be discussed further with regard to specific entries in the tables that are used in the preferred embodiment. These entries are specified as "Related Figures" in Table 2. It should be apparent to one skilled in the art that the specific arrangement of bits and the number of bits may vary between embodiments. In other embodiments, the actual page size for example may be different, or the physical address size may be different. FIGS. 5-9 are provided in order to comply with the full disclosure and provide an exemplary arrangement for implementing dual page size and extended physical addressing. Many of the bits and bit names have been used for Intel's i486™ micro processor, and are described in detail in the "i486™ Microprogrammer's Reference Manual", particularly at Section 5.3.3, available from Intel Corporation of Santa Clara, Calif.

Reference is made to FIG. 5 which illustrates the format of the registers and table entries pertinent to operation in Mode A. Mode A is the original and default type of the X86 family. It maps a standard 4K-byte page within a 32-bit physical address space. The look-up tables include both a page directory and a page table directory.

The page directory entry is 4 bytes wide and thus requires 30 bits to specify a unique address. The first 20 bits point to

Reference is made to FIG. 7 which is a depiction of the control register and the page directory entry for a large page size and a normal physical address space. The control register entries shown in FIG. 7 implement the table structure illustrated in FIG. 3. Type 2 results in a Translation Lookaside Buffer (TLB) entry that maps a 4 Mbyte page within a 32-bit physical address space. The look-up involves only a page directory entry as illustrated in FIG. 3.

The page directory entry (PDE) is retrieved and deemed valid in exactly the same way as the PDE of type 1. In this type of look-up the PDE.PS bit will be found set, thus no table look-up will take place. From this point on, the bit checking of the PDE will be virtually identical to the bit checking of the page table entry (PTE) of type 1. The only difference is that the U and W statuses are the PDE.U and PDE.W bits themselves.

Reference is made to FIG. 8 which illustrates the format for the control register and the table entries, particularly the page directory pointer format and the page directory entry format. Reference is also made to FIG. 2, which is implemented with the formats shown in FIG. 8.

The type 3 and extended addressing Mode C results in a TLB entry able to map a 4K-byte page within a 36-bit physical address space. The look-up uses a page directory pointer, a page directory entry, and a page table entry. There are four page directory pointers (PDPTR (3-0)), located physically in registers within the P-unit. The page directory pointers are stored there by microcode when CR4.PAE is set, as part of a write to control register 3. The particular page directory pointer to look-up is selected by bits 31:30 of the linear address. The P bit is checked, to see that it points to a valid page directory, and then the RSV bits of the pointer are verified to be zero. If the RSV bits are not all zero a page fault is returned.

5,617,554

11

Within this Mode C, the PDE is eight bytes wide and thus requires 33 bits to specify a unique address. The first 24 bits point to the page directory and come from the selected page directory pointer bits 35:12. The particular page directory entry is selected from the linear address bits 29:21. Within one 4K-byte page, 512 eight byte entries can fit. The bit checks of the PDE are handled in the same fashion as those of the PDE for type 1, with the main difference being that there are 28 more RSV bits.

The page table entry (PTE) is also eight bytes wide and accessed similarly to the PDE. The only difference is that the page table is at bits 35:12 of the PDE in the offset is in the linear address bits 20:12. Similarly, the bit checks of the PTE are handled similarly as the PTE of type 1, with the main difference being that there are 28 more RSV bits. The TLB entry for Mode C occupies the full Translation Lookaside Buffer of 36 bits.

Reference is made to FIG. 9 which depicts the format of the control register and the entries into the page directories and the page directory pointers. These register formats implement the embodiment of FIG. 4, for the extended physical addressing and a large page size. Specifically, type 4 and mode C results in a TLB entry able to map a 2M-byte page within a 36-bit physical address space. The look-up involves selection of a pointer from the page directory pointers, which select a particular page directory as illustrated in FIG. 4.

The PDE is retrieved and deemed valid in exactly the same way as the PDE of type 3. In this type of look-up the PDE.PS bit will be found set, thus no table look-up will take place. From this point on the bit checking of the PDE will be virtually identical to the bit checking of the PTE of type 3. The main difference is that the U and W statuses are the PDE.U and the PDE.W bits themselves.

In order to implement extended physical addressing, it should be apparent that the buses and particularly the TLB and the TLB bus be able to support the size of the extended physical address. In the preferred embodiment, the size of the physical address is 36 bits, and therefore the TLB and the TLB bus are implemented to store and communicate this size physical address.

Reference is made to FIG. 10, which is an illustration of the relationship between a microprocessor 100 and a computer memory 102. The microprocessor includes a control unit 104 which has numerous registers provided therein including a CR3 register 106 and a CR4 register 108. The microprocessor 100 also includes a page unit 110 which includes logic and additional circuitry including hardware, PLA programming sequencing control, and pointer registers 112. The page unit 110, sometimes referred to as the "P-unit," controls the address translation as described above. For example, the P-unit controls all table manipulations and directs transfers with main memory for missing TLB entries.

The computer memory 102 includes a main memory 114 that may, for example, be a Random Access Memory (RAM) that is easily accessible by a user. The computer memory 102 also may include one or more secondary memories such as a disk memory 116. A secondary memory typically requires more time for each memory access than the main memory 114. As described above, the page directories and the page tables are located in computer memory 102. The directory pointer table is also located in computer memory 102, but additionally its same entries are located within the page unit 110, specifically within the pointer registers 112. Loading the pointer registers 112 from the computer memory 102 occurs in the preferred embodiment whenever the CR3

12

register 106 is loaded. Because the pointers are in the registers 112, within the page unit 110 access time is substantially reduced and therefore performance is increased.

The term "physical memory" defines the physical address space seen by the operating system. The size of the physical address space is usually larger than the available main memory, but data at a specific physical address must be accessed within the main memory. A page validity bit is associated with each page table entry to indicate whether or not the page is in main memory. If the page validity bit indicates that a requested page is not in main memory, then an operating system exception is generated and the data is transferred from secondary storage to main memory. Then, the page validity bit is set to indicate that the page is now in main memory.

Reference is made to the flow chart of FIGS. 11 and 12, which together illustrate operation of page translation in the preferred embodiment. The flow chart of FIGS. 11 and 12 shows translation for a selectable memory size, and a selectable page size. Following the initialization 120 step at the top of the chart, a memory size is selected in a decision box 121. If the memory size is selected to be "small", then the table entries are formatted accordingly and set up with a size of 32 bits, as illustrated in an operation box 122. However, if a "large" memory size is selected, then the pointers and the table entries are set up to a 64-bit size, as illustrated in operation box 124. From both operation boxes 122 and 124 control flows to decision box 126. At decision box 126, a wait loop is performed until a new task is created. Whenever a new task is created, the number of pages and page sizes for each table are selected in operation box 128. Following operation box 128, each table is set up in operation box 129. Table set up includes setting the page bit based upon the selected page size.

As part of the operation box 129, the page size is selected as illustrated in a decision box 130. If a "small" size is selected, then the page bit remains clear in the page directory entry as illustrated in a box 131. directory entries that are created have their page bit set, as illustrated in operation box 132. Upon completion of table set up in operation box 129, it is determined if all tables have been set up at decision box 134. If all the tables have not been set up for the new task, then operation boxes 130-132 are repeated for each table until all tables have been set up for the task.

After all the tables for the new task have been set up, a wait loop is entered at decision box 136 until the task is selected. When the task is selected, it is determined whether or not the PAE bit is set as illustrated in decision box 138. If the PAE bit is not set, then the processor is ready to perform memory references. If the PAE bit is set, then the pointer registers are loaded into the page unit 110 as illustrated in operation box 140 and the processor is ready to perform memory references. Because the set up is now complete, the processor stands ready to request a memory reference as illustrated in a decision box 142. Moving the reference from FIG. 11 to FIG. 12 after the memory reference has been requested, the selected size of memory is reviewed as illustrated in a decision box 142. If a small size was previously selected, then the contents of the control register are applied to select a particular page directory, as illustrated in the operation box 144. However, if a large memory size is selected, then, as illustrated in an operation box 146, a pointer is selected by applying a pointer field of the linear address. Subsequently, as illustrated in a box 148, a page directory is selected by applying the selected pointer. After the page directory has been selected then, as illustrated

5,617,554

13

in a box 150 the directory field of the linear address is applied to select a directory entry. Next, as illustrated in a decision box 152, the directory entry is checked to see whether or not the page bit has been set for a large page size. If a large page size has been selected, then the page frame is selected directly by applying the directory entry, as illustrated in a box 154. However if the page bit was not set for a large page size (i.e., a small page size has been selected) then, as illustrated in a box 156 a page table is selected using the directory entry. Next, as illustrated in a box 158, the page field of the linear address is applied to the page table to select a page frame. After a page frame has been selected, either in the box 158 or the box 154, then as illustrated in the box 160 the offset field of the linear address is applied to select an operand in the page frame.

The invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiment is to be considered in all respects only as illustrative and not restrictive and the scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing descriptions. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed is:

1. A processor generating linear addresses, said processor comprising: a control unit having stored therein an indication in one of a plurality of states; and a paging unit coupled to said control unit to receive said indication, said paging unit translating said linear addresses into a physical address for accessing a physical address space, said paging unit simultaneously supporting paging using at least a first and a second page frame size while said indication is in a first of said plurality of states, said paging unit supporting paging using only one page frame size while said indication is in a second of said plurality of states.

2. The processor of claim 1, wherein said second page frame size is larger than said first page frame size.

3. The processor of claim 2, wherein said first page frame size is 4K and said second page frame size is 4M.

4. The processor of claim 1, wherein said physical address space has a different number of addressable locations while said indication is in said first state than while said indication is in said second state.

5. The processor of claim 4, wherein each of said linear addresses includes N-bits, said physical address space has no more than 2^N addressable locations while said indication is in said second state, and said physical address space has more than 2^N addressable locations while said indication is in said first state.

6. An address translator for physical memory, said address translator translating a linear address having no more than N bits into a physical address, said address translator comprising:

memory size selection means for selecting a physical address size to be a first address size having no more than said number of N bits or a second address size designated by a number greater than said N bits;

one or more page directories, each having a group of page directory entries, said linear address having a page directory field for selecting a page directory entry;

a plurality of page tables, each having a group of page table entries, said linear address having a format, said format selected from a plurality of predetermined formats, at least one of said plurality of predetermined formats having a page table field for selecting a page table entry;

means, responsive to said memory size selection means, for formatting the page table entries to a first page table

14

entry size if the first address size has been selected or to a second page table entry size if the second address size has been selected, said second page table entry size being larger than the first page table entry size; and

paging means for simultaneously supporting paging using at least a first and a second page frame size while one or more control bits are in a first state, said paging means supporting paging using only one page frame size while said one or more control bits are in a second state.

7. The address translator of claim 6 further comprising: page size selection means for selecting a page size to be a first page size or a second page size, said second page size being larger than the first, said page size selection means including a flag in each page directory entry that is indicative of the selected size of one or more corresponding pages.

8. The address translator of claim 8, further comprising: means, responsive to said memory size selection means, for formatting the page directory entries to a first entry size if the first address size has been selected or to a second entry size if the second address size has been selected, said second entry: size being larger than the first entry size.

9. The address translator of claims 8 further comprising: page size selection means for selecting a page size to be a first page size or a second page size, said second page size being larger than the first, said page size selection means including a flag in each page directory entry that is indicative of the selected size of one or more corresponding pages.

10. The address translator of claim 6 further comprising: a directory pointer table including a group of directory pointer table entries if said second address size has been selected, said format of said linear address having a pointer selection field for selecting one of said directory pointer table entries if said second address size has been selected.

11. The address translator of claim 10 further comprising: a control register containing a value, said value for selecting a page directory from said one or more page directories if the first address size has been selected, said value for selecting said directory pointer table if the second address size has been selected.

12. The address translator of claim 10, further comprising a plurality of registers for holding said directory pointer table entries.

13. The address translator of claim 12, wherein the registers are dedicated for holding said directory pointer table entries.

14. A method for use by a processor to translate a linear address having a size of no more than N bits into a physical address having a size of a first address size designated by a number no more than N bits or a second address size designated by a number greater than N bits, said method using a plurality of tables including one or more page directories with page directory entries and at least one page table with page table entries, said method comprising the computer implemented steps of:

said processor altering a physical address mode indicator, said physical address mode indicator identifying said physical address size to be the first address size or the second address size;

if the first address size has been selected, then said processor setting the page directory entries and the page table entries to a first entry size;

5,617,554

15

if the second address size has been selected, then said processor setting the page directory entries and the page table entries to a second entry size that is larger than the first entry size; and

said processor translating said linear address into said physical address using those of said plurality of tables having a corresponding field in said linear address, said processor simultaneously supporting paging using at least a first and a second page size while one or more control bits are in a first state, said processor supporting paging using only one page size while said one or more control bits are in a second state.

15. The address translation method of claim 14, wherein said first page size is 4K and said second page size is 4M.

16. The address translation method of claim 14, further comprising the steps of:

providing a flag in each page directory entry;

for each of said page directory entries, performing the steps of:

selecting a page size to be said first page size or said second page size, said second page size being larger than the first page size; and

altering said flag to indicate said page size.

17. The address translation method of claim 16 further comprising the steps of:

if the first address size has been selected, then applying the contents of a control register to select one of said one or more page directories;

if the second address size has been selected, then applying a pointer field of the linear address to select a pointer from a directory pointer table, and applying said selected pointer to select one of said one or more page directories.

18. The address translation method of claim 17, further comprising the steps of:

applying a directory field from the linear address to select one of said page directory entries as a selected page directory entry;

testing said flag in the selected page directory entry; and if said flag in said selected page directory indicates the second page size has been selected, then applying the directory entry to select a page frame;

if said flag in said selected page directory indicates the first page size has been selected, then applying the page directory entry to select a page table, and applying a page table field from the linear address to select a page frame from the page table; and

applying an offset field from the linear address to the selected page frame to select.

19. A computer system comprising:

a processor including:

a control unit having stored therein an indication in one of a plurality of states, and

a paging unit coupled to said control unit to receive said indication, said paging unit simultaneously supporting paging using at least a first and a second page frame size while said indication is in a first of said plurality of states, said paging unit supporting paging using only one page frame size while said indication is in a second of said plurality of states; and

a memory coupled to said processor, said memory having stored therein a plurality of page tables for use by said paging unit to simultaneously support paging using at least said first and said second page frame sizes.

16

20. The processor of claim 19, wherein said second page frame size is larger than said first page frame size.

21. The processor of claim 20, wherein said first page frame size is 4K and said second page frame size is 4M.

22. A method for use by a processor to translate a linear address to a physical address in a physical address space, said processor including a control unit coupled to a paging unit, said method comprising the steps of:

(a) said processor receiving a first instruction;

(b) in response to said first instruction, said processor storing an indication in one of a plurality of states in said control unit;

(c) said paging unit translating said linear address into said physical address said paging unit simultaneously supporting paging using at least a first and a second page frame size while said indication is in a first of said plurality of states, said paging unit supporting paging using only one page frame size while said indication is in a second of said plurality of states.

23. The method of claim 22 wherein said second page frame size is larger than said first page frame size.

24. The method of claim 23, wherein said first page frame size is 4K and said second page frame size is 4M.

25. The method of claim 22, wherein said physical address space has a different number of addressable locations while said indication is in said first of said plurality of states than while said indication is in said second of said plurality of states.

26. The method of claim 25, wherein said linear address includes N-bits, said physical address space has no more than 2^N addressable locations while said indication is in said second of said plurality of states, and said physical address space has more than 2^N addressable locations while said indication is in said first of said plurality of states.

27. An address translation apparatus for use by a processor to translate a linear address having no more than N bits into a physical address, said linear address having a first field and a second field, said address translation apparatus comprising:

means for executing one or more instructions on said processor to store a physical address mode indicator, said physical address mode indicator identifying which of a first physical address size and a second physical address size will be utilized by said processor, said first physical address size having no more than 2^N locations that can be addressed, said second physical address size having more than 2^N locations that can be addressed;

means for formatting page directory entries in one or more page directories and page table entries in a plurality of page tables to be compatible with the physical address size identified by said physical address mode indicator;

means for applying both said first field and a value to select a page directory entry as a selected page directory entry if said first physical address size is identified by said physical address mode indicator, wherein said first field is applied as a directory field and said value is stored in said processor;

means for applying both a first portion of said first field and said value to select one of a plurality of directory pointer table entries as a selected directory pointer table entry, selecting one of a plurality of page directories as said selected page directory using said selected directory pointer table entry, and applying a second portion of said first field as a directory field to select a page directory entry as a selected page directory entry from

5,617,554

17

said selected page directory if said second physical address size is identified by said physical address mode indicator, wherein said first portion of said first field is applied as a pointer field; and

means for applying said selected page directory entry and said second field to provide a physical address, wherein at least a first and a second page frame size is supported while a set of control bits is in a first state and only one page frame size is supported while said set of control bits is in a second state.

28. The address translation apparatus of claim **27**, wherein said means for applying said selected page directory entry and said second field to provide a physical address further

18

includes applying a first portion of said second field and a second portion of said second field if said selected page directory entry indicates said physical address is located in a page frame having said first page frame size, and if said selected page directory entry indicates said physical address is located in a page frame having said second page frame size, said second field is applied as an offset field, wherein said first portion of said second field is applied as a page table field and said second portion of said second field is applied as an offset field.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,617,554
DATED : April 1, 1997
INVENTOR(S) : Alpert et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In column 12 at line 39 insert --However, if a "large" page size is selected, then the page-- following "**131**." and prior to "directory"

In column 14 at line 25 delete "claims" and insert --claim--

In column 15 at line 43 insert --page-- following "the"

In column 15 at line 51 insert --an operand-- following "select" and prior to "."

Signed and Sealed this
Twenty-second Day of July, 1997



Attest:

BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

EXHIBIT 3



US005802605A

United States Patent [19]

Alpert et al.

[11] **Patent Number:** 5,802,605[45] **Date of Patent:** *Sep. 1, 1998[54] **PHYSICAL ADDRESS SIZE SELECTION AND PAGE SIZE SELECTION IN AN ADDRESS TRANSLATOR**[75] **Inventors:** Donald B. Alpert, Santa Clara; Kenneth D. Shoemaker, Saratoga, both of Calif.; Kevin C. Kahn, Portland; Konrad K. Lai, Aloha, both of Oreg.[73] **Assignee:** Intel Corporation, Santa Clara, Calif.[*] **Notice:** The term of this patent shall not extend beyond the expiration date of Pat. No. 5,617,554.[21] **Appl. No.:** 756,184[22] **Filed:** Nov. 25, 1996**Related U.S. Application Data**

[63] Continuation of Ser. No. 372,805, Dec. 23, 1994, Pat. No. 5,617,554, which is a continuation of Ser. No. 832,944, Feb. 10, 1992, abandoned.

[51] **Int. Cl.⁶** G06F 12/10[52] **U.S. Cl.** 711/208; 711/212[58] **Field of Search** 395/421.02, 416, 395/418[56] **References Cited****U.S. PATENT DOCUMENTS**

| | | | |
|-----------|---------|-----------------|---------|
| 4,340,932 | 7/1982 | Bakula et al. | 395/402 |
| 4,432,053 | 2/1984 | Gaither et al. | 395/416 |
| 4,591,972 | 5/1986 | Guyer et al. | 395/569 |
| 4,654,777 | 3/1987 | Nakamura | 395/416 |
| 4,669,043 | 5/1987 | Kaplinsky | 395/403 |
| 4,679,140 | 7/1987 | Gotou et al. | 395/775 |
| 4,758,946 | 7/1988 | Shar et al. | 395/416 |
| 4,763,250 | 8/1988 | Keshlear et al. | 395/418 |
| 4,792,897 | 12/1988 | Gotou et al. | 395/417 |
| 4,835,734 | 5/1989 | Kodaira et al. | 395/419 |
| 4,972,338 | 11/1990 | Crawford et al. | 395/416 |
| 4,979,098 | 12/1990 | Baum et al. | 395/418 |
| 5,023,777 | 6/1991 | Sawamoto | 395/402 |
| 5,263,140 | 11/1993 | Riordan | 711/207 |
| 5,475,827 | 12/1995 | Lee et al. | 711/207 |

FOREIGN PATENT DOCUMENTS

0113240 12/1983 European Pat. Off. .
 1595740 5/1978 United Kingdom .
 2127994 6/1983 United Kingdom .

OTHER PUBLICATIONS

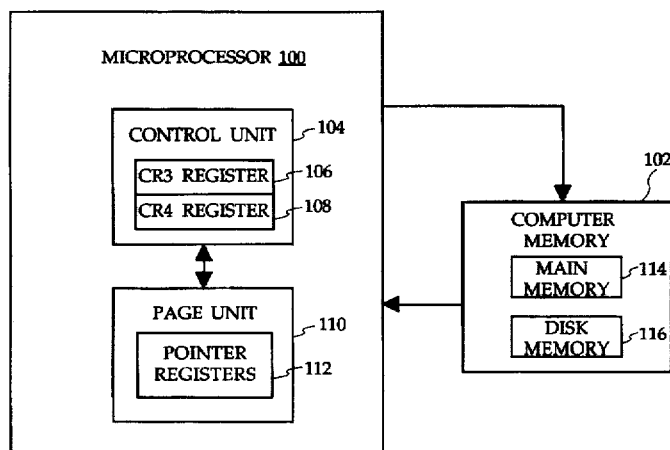
i860™ *Microprocessor Family Programmer's Reference Manual*, Intel Corporation Literature Sales, Chapter 4, pp. 1-13 (1991).

Patterson, David A. and Hennessey, John L. *Computer Architecture: A Quantative Approach*, Morgan Kaufman Publishers, Inc. pp. 432-485, (1990).

(List continued on next page.)

Primary Examiner—Eddie P. Chan*Assistant Examiner*—Reginald G. Bragdon*Attorney, Agent, or Firm*—Blakely, Sokoloff, Taylor & Zafman[57] **ABSTRACT**

An address translator and a method for translating a linear address into a physical address for memory management in a computer is described herein. Different memory sizes, and different page sizes can be selected. The address translator can translate from a standard 32-bit linear address for compatibility with previous 32-bit architectures, and can also translate to a physical memory size with a larger physical address than linear address; i.e., greater than 32 bits (e.g. 36 bits and up), with no increase in access time. The address translator translates a linear address that includes an offset and a plurality of fields used to select entries in a plurality of tables. The format of the linear address into fields is dependent upon the selected memory size and the selected page size. For a large memory size, the tables include a directory pointer table that includes a group of directory pointers, a plurality of page table directories each of which includes a group of page directory entries, and a plurality of page tables each of which includes a group of page table entries. The size of the entries in the tables is dependent upon the selected memory size. The contents of the tables are stored in memory, and furthermore the pointer table is stored in both main memory and in dedicated pointer table registers.

18 Claims, 12 Drawing Sheets

5,802,605

Page 2

OTHER PUBLICATIONS

Nelson, Ross P. *The 80386 Book*, Microsoft Press, Chapter 6, pp. 125–134, (1988).

i486™ Processor Programmer's Reference Manual, Intel Corporation Literature Sales, Chapter 5, pp. 1–25 (1990).

i860™ XP Microprocessor Data Book, Intel Corporation Literature Sales, Chapter 2, pp. 21–27, (May 1991).

SUN Microsystems, *The SPARC™ Architecture Manual Version 8*, Sun Microsystems, Inc., pp. 237–255, (Dec. 11, 1990).

U.S. Patent

Sep. 1, 1998

Sheet 1 of 12

5,802,605

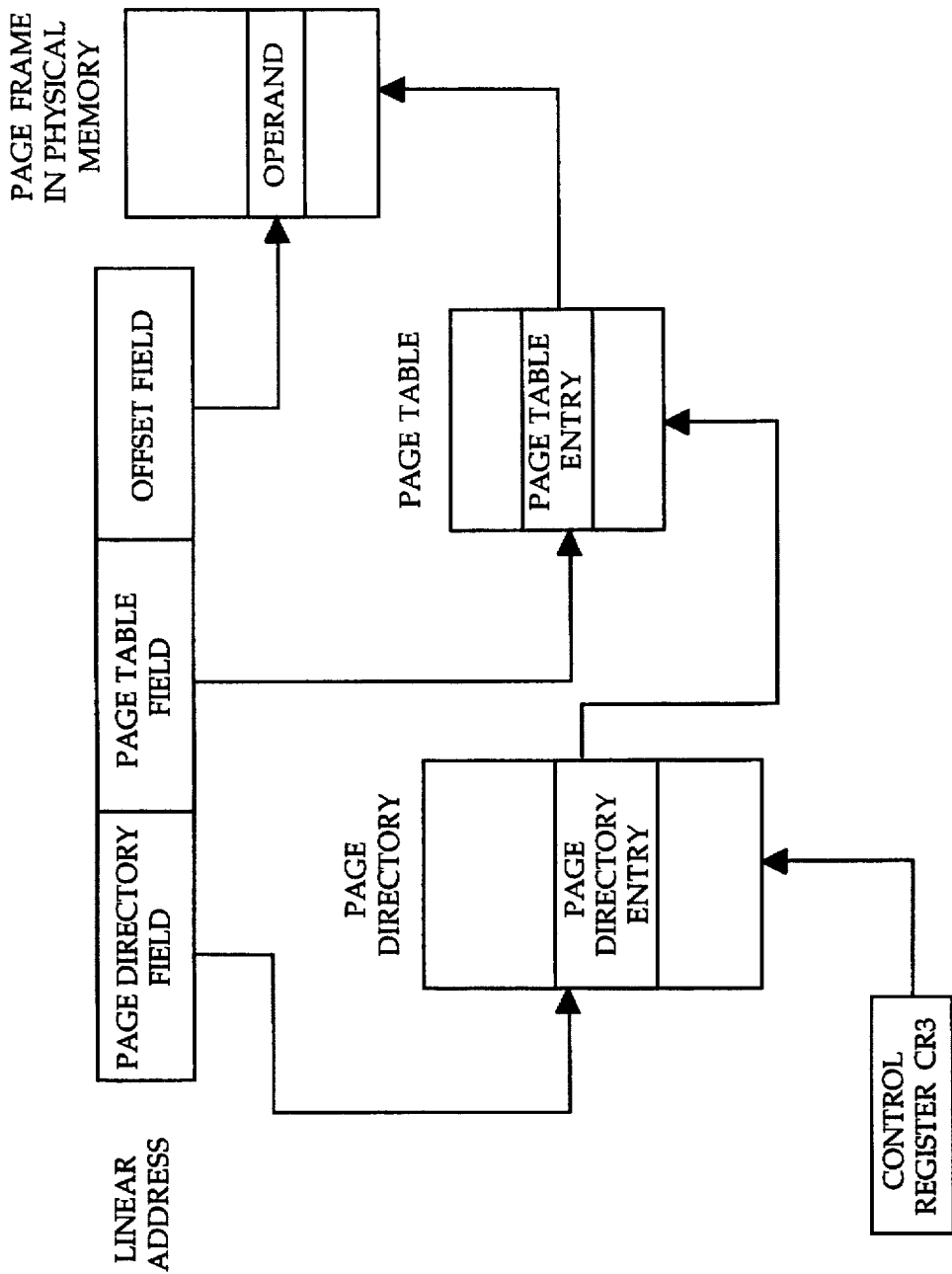


FIG. 1 (PRIOR ART)

U.S. Patent

Sep. 1, 1998

Sheet 2 of 12

5,802,605

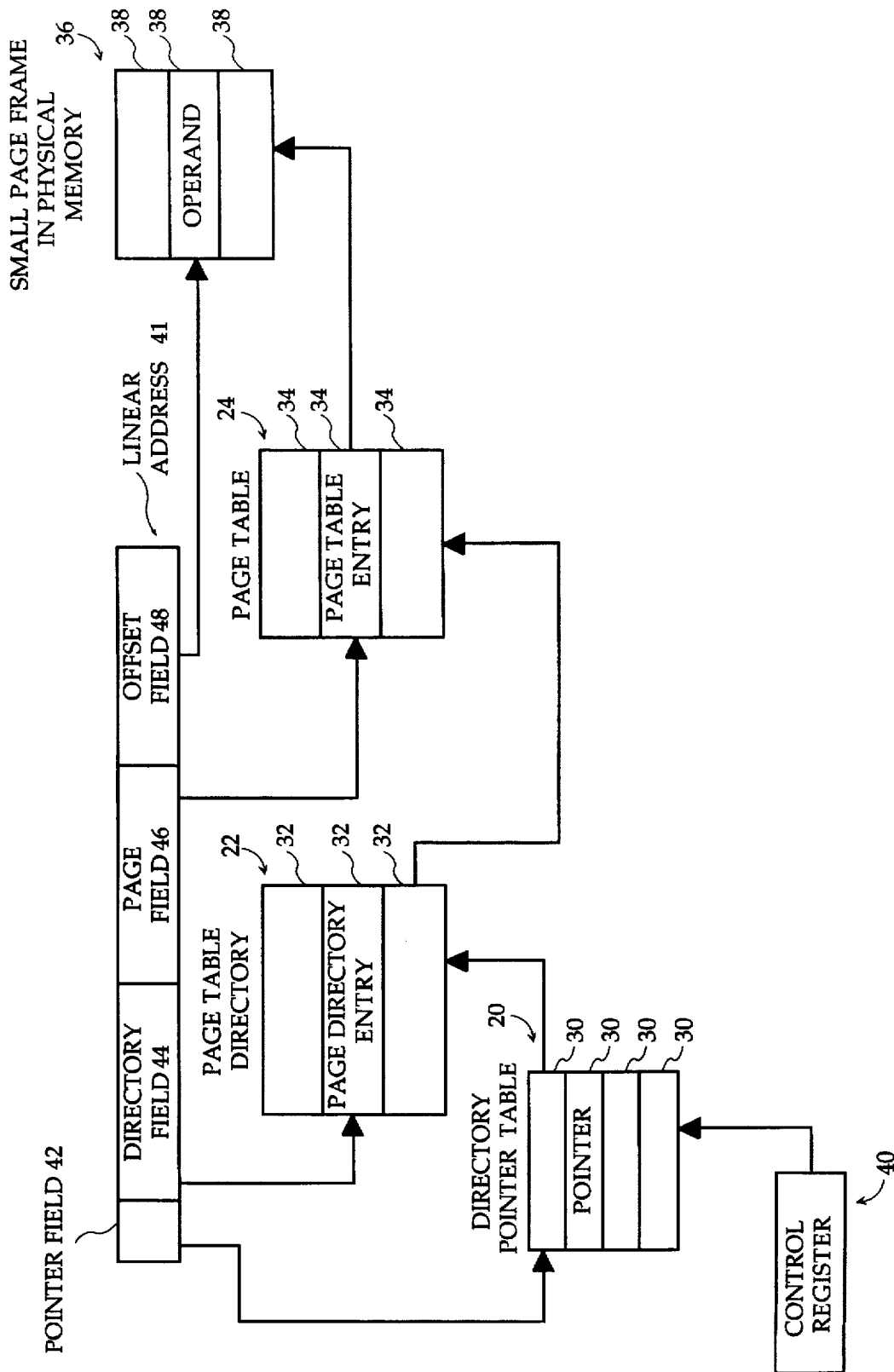


FIG. 2

U.S. Patent

Sep. 1, 1998

Sheet 3 of 12

5,802,605

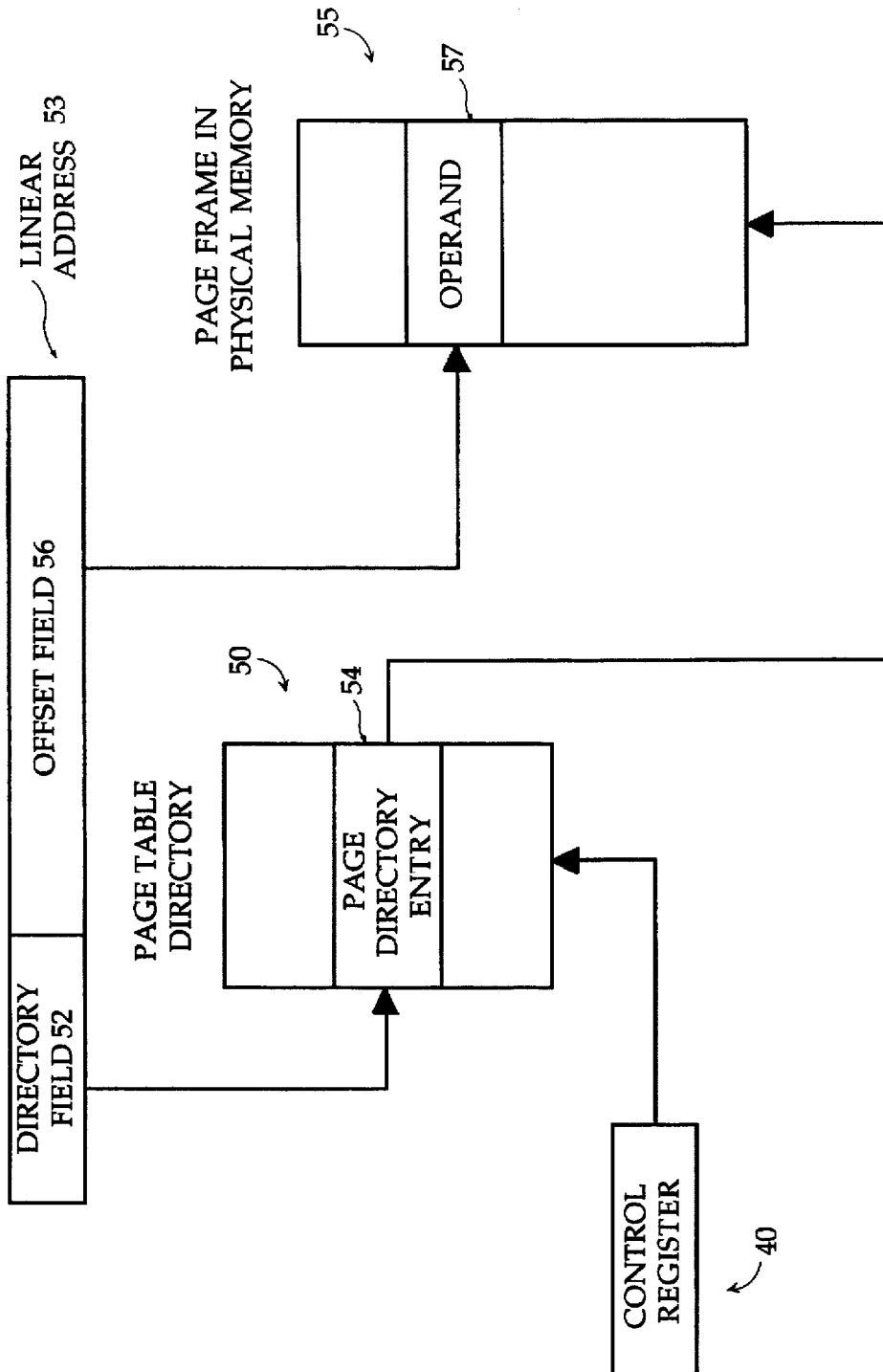


FIG. 3

U.S. Patent

Sep. 1, 1998

Sheet 4 of 12

5,802,605

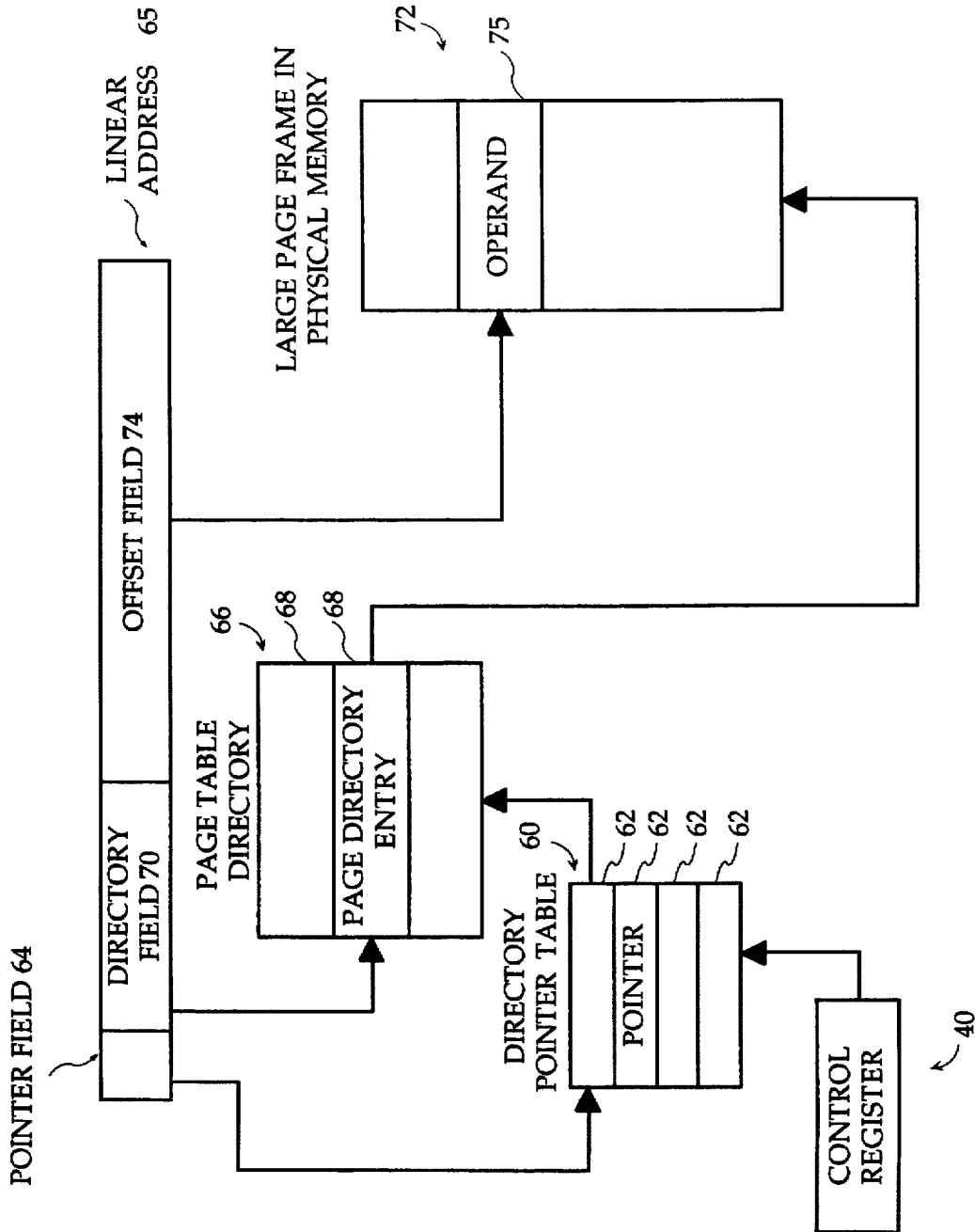


FIG. 4

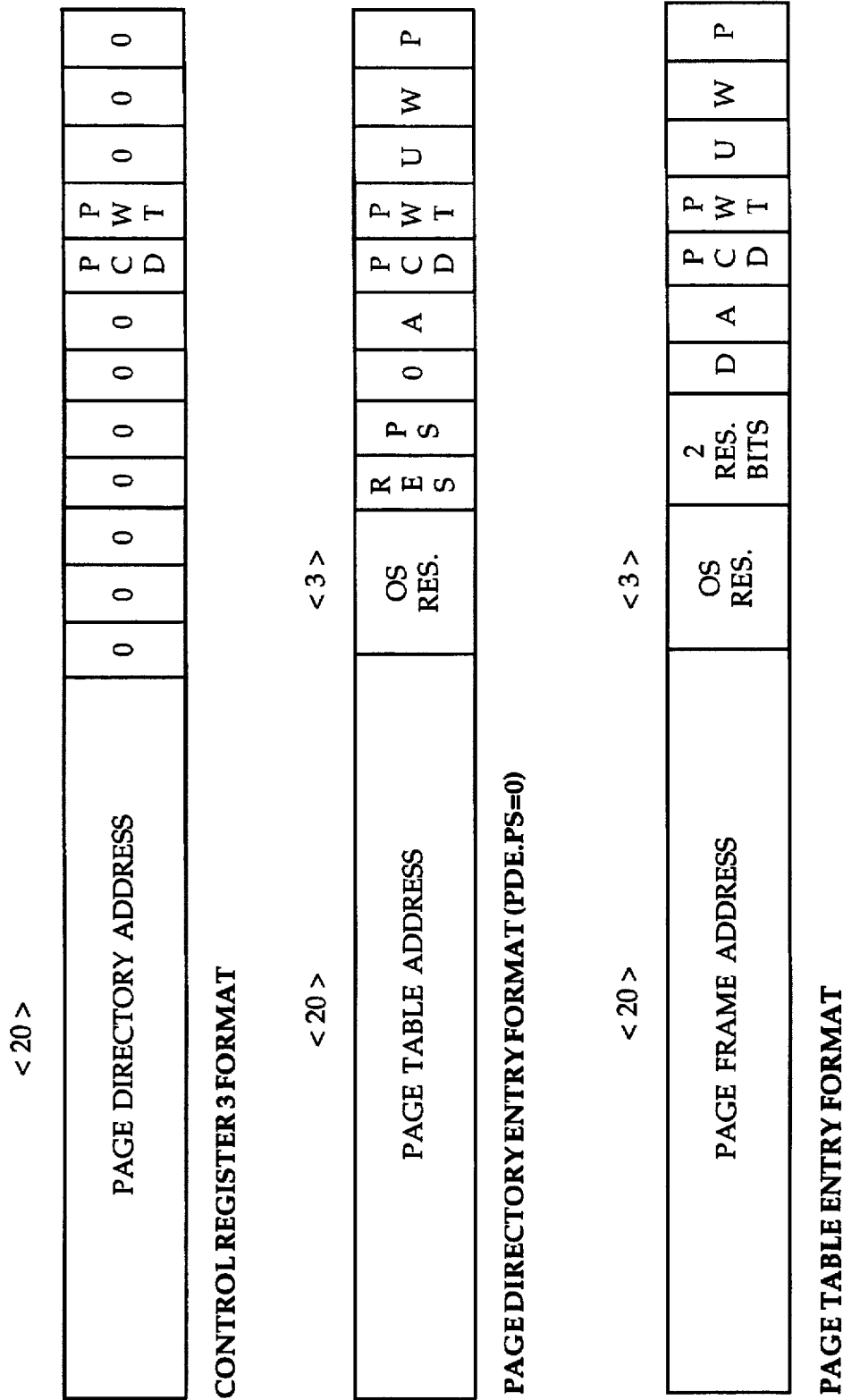


FIG. 6

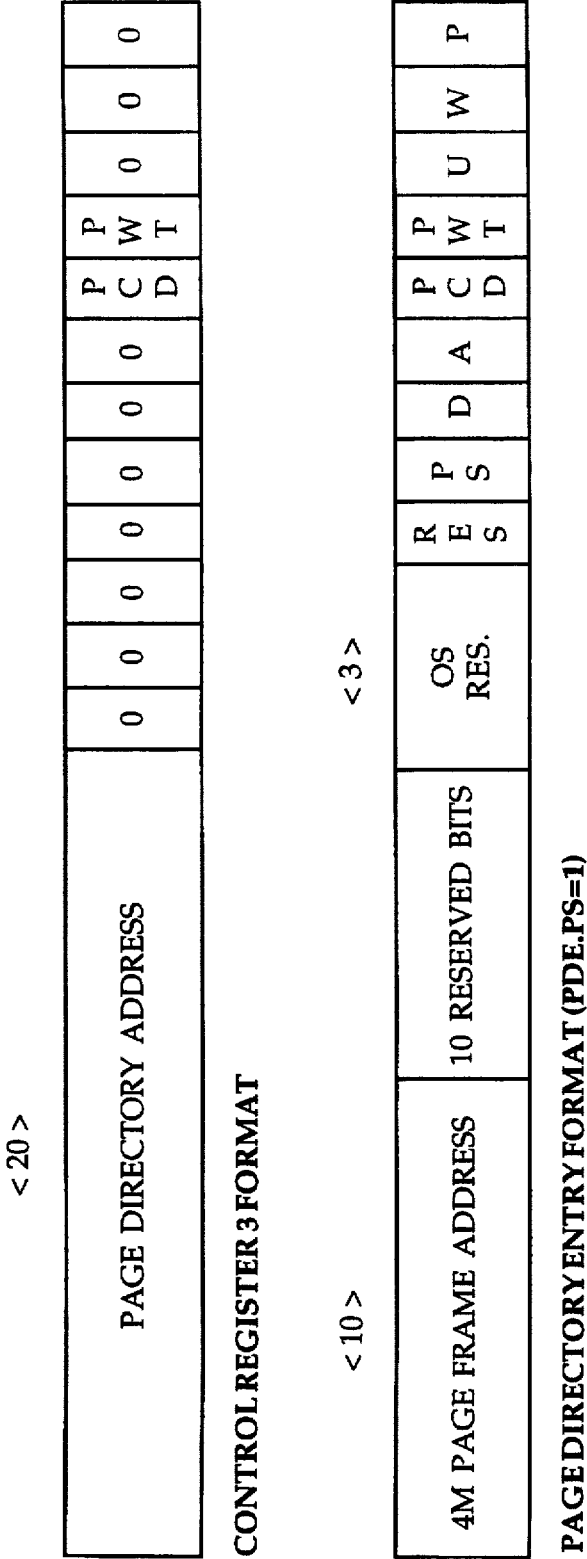


FIG. 7

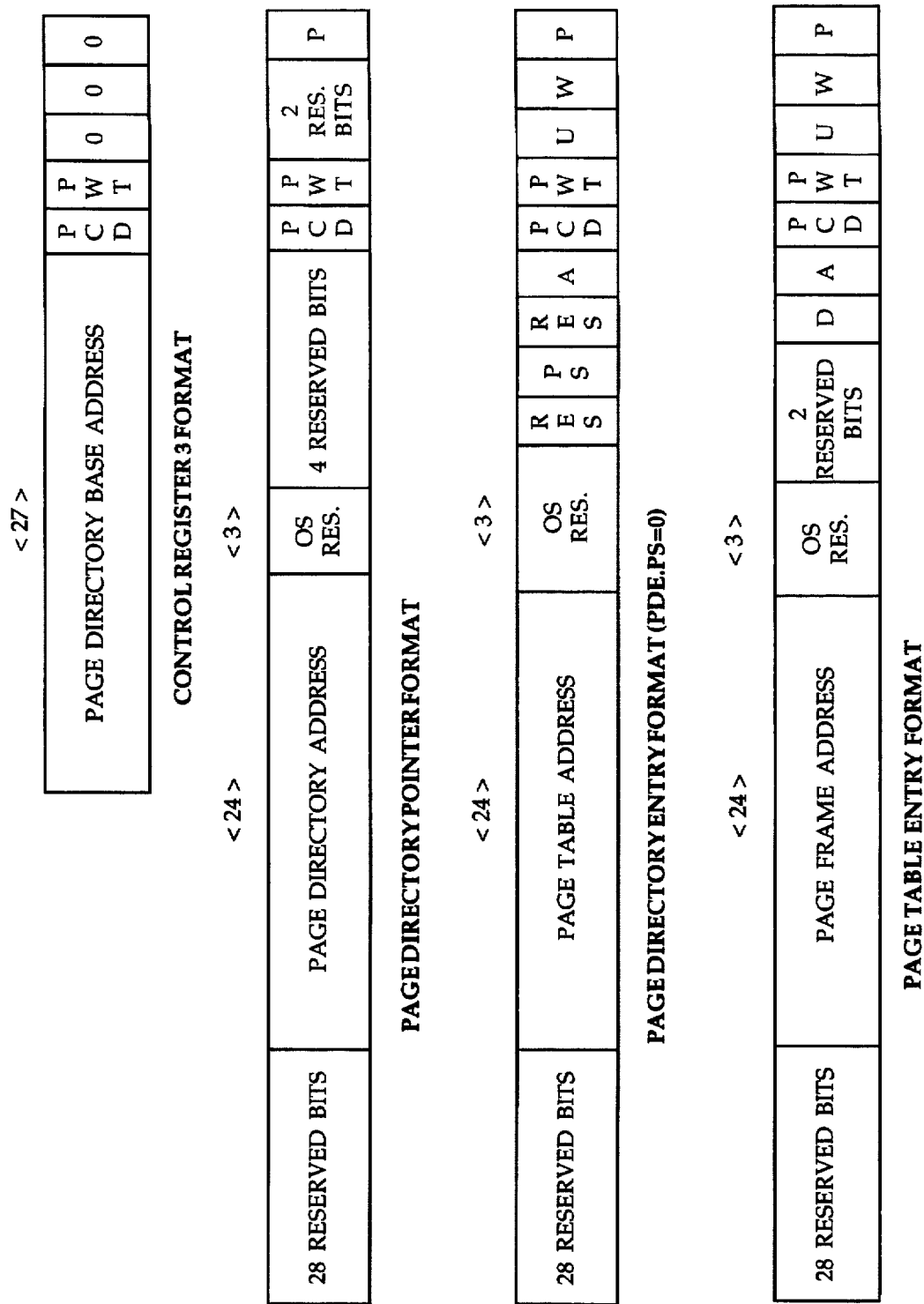


FIG. 8

< 27 >

| | | | | | | | | | |
|-----------------------------|--|--|--|--|---|---|---|---|---|
| PAGE DIRECTORY BASE ADDRESS | | | | | P | P | 0 | 0 | 0 |
| | | | | | C | W | | | |
| | | | | | D | T | | | |

CONTROL REGISTER 3 FORMAT

< 24 >

| | | | | | | | |
|------------------|------------------------|------------|-----------------|-------------|-------------|-------------------|---|
| 28 RESERVED BITS | PAGE DIRECTORY ADDRESS | OS RES. | 4 RESERVED BITS | P C D | P W T | 2 RES. BITS | P |
| | | | | | | | |

PAGE DIRECTORY POINTER FORMAT

< 24 >

| | | | | | | | | | | | | |
|------------------|-----------------------------|--------------------|------------|-------------|--------|---|---|-------------|-------------|---|---|---|
| 28 RESERVED BITS | 2M PAGE FRAME ADDRESS | 9 RESERVED BITS | OS RES. | R E S | P S | D | A | P C D | P W T | U | W | P |
| | | | | | | | | | | | | |

PAGE DIRECTORY ENTRY FORMAT (PDE.PS=1)

FIG. 9

U.S. Patent

Sep. 1, 1998

Sheet 10 of 12

5,802,605

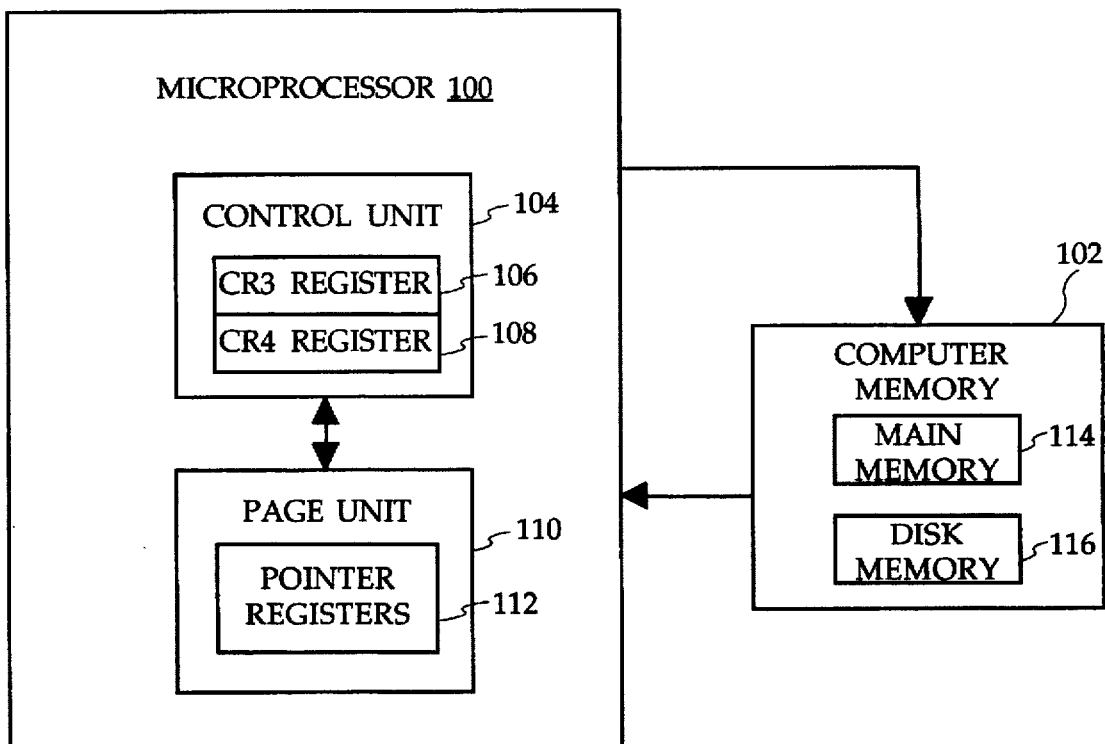


FIG. 10

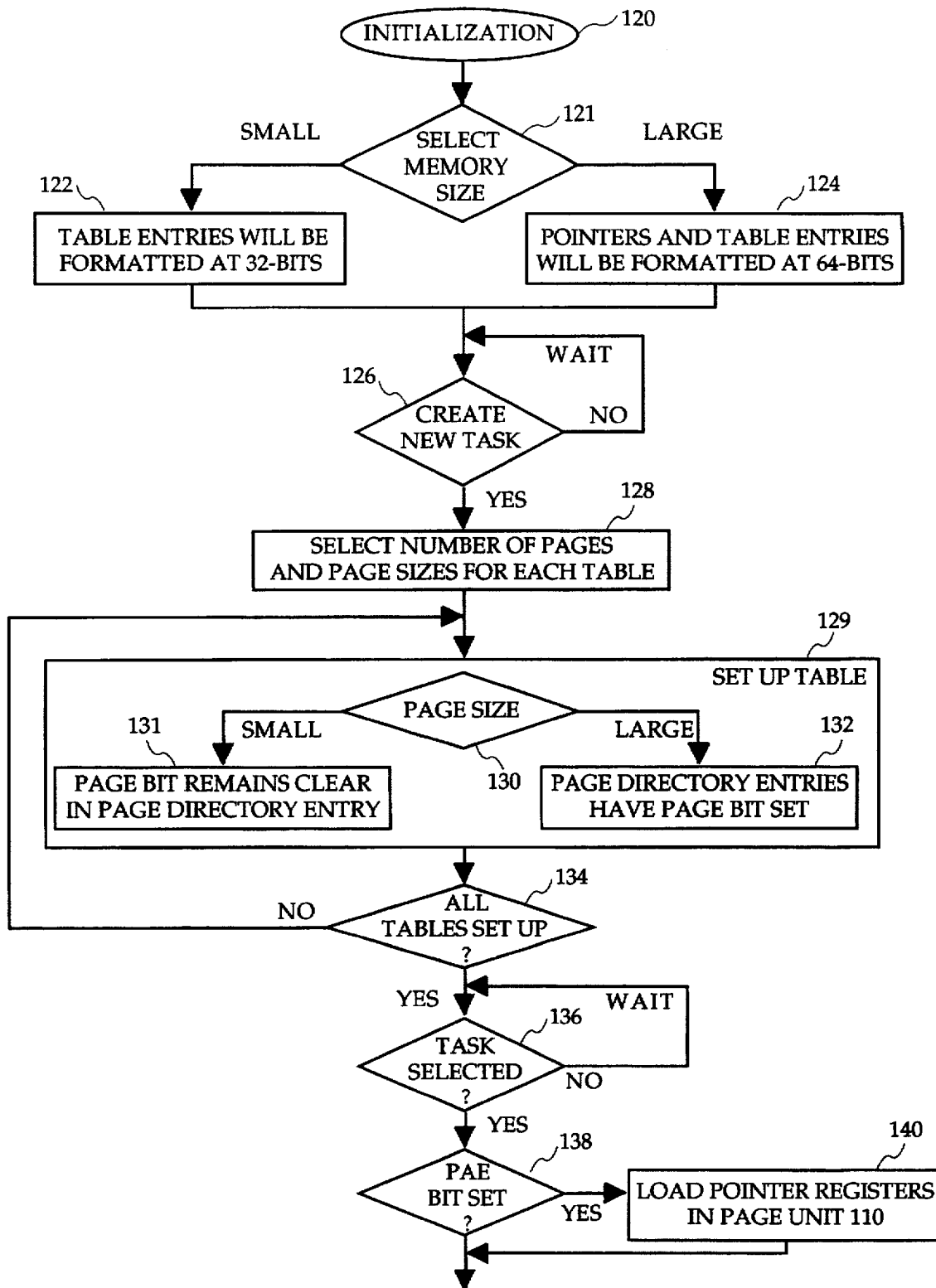


FIG. 11

READY TO PERFORM MEMORY REFERENCE

U.S. Patent

Sep. 1, 1998

Sheet 12 of 12

5,802,605

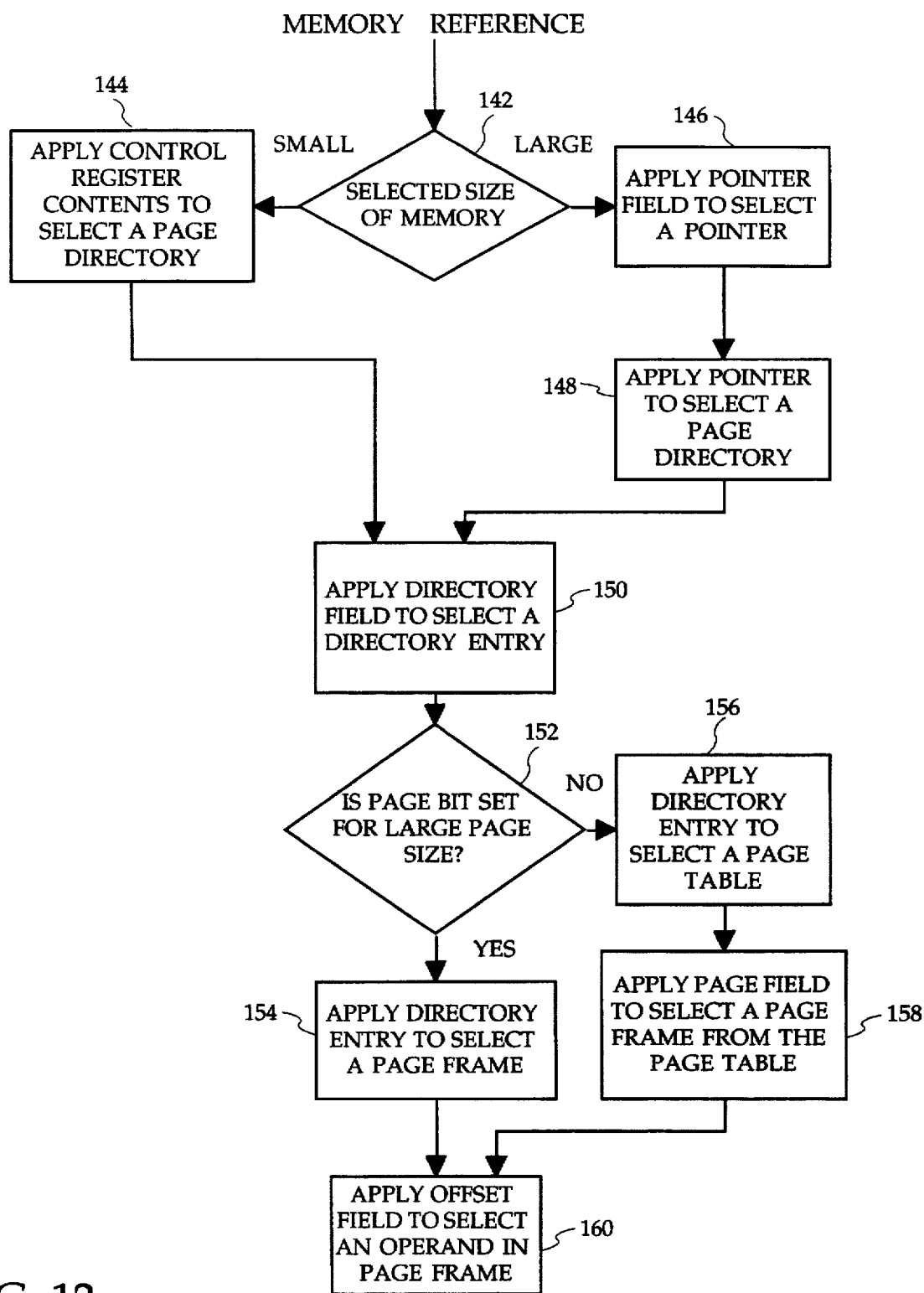


FIG. 12

5,802,605

1

PHYSICAL ADDRESS SIZE SELECTION AND PAGE SIZE SELECTION IN AN ADDRESS TRANSLATOR

This is a continuation of application Ser. No. 08/372,805, filed on Dec. 23, 1994, now U.S. Pat. No. 5,617,554, which is a continuation of application Ser. No. 07/832,944, filed on Feb. 10, 1992, abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to computer memory management methods and apparatus for translating between a virtual address and a physical address stored in a plurality of pages in main memory.

2. Description of Related Art

A computer typically includes main memory and secondary memory. A microprocessor may be included to control the writing and reading of these memory elements. Main memory, constructed of RAM (Random Access Memory) chips is much faster than secondary memory such as a hard disk drive, but main memory is typically much more expensive per memory element. Thus, computers are usually designed with a main memory limited in size and a much larger secondary memory. Microprocessors designed to use virtual memory effectively extend main memory space into secondary memory space. Typically, virtual memory microprocessors use techniques such as paging or segmentation, or both, to simulate a larger memory.

Virtual memory provides many advantages. It is possible to run applications that require more RAM (Random Access Memory) than is actually available. Furthermore, it is possible for more than one application to run at the same time. Virtual memory gives an applications programmer a view of more main memory than actually exists. To a programmer, virtual memory appears as one contiguous block of main memory. An applications programmer seeking storage space does not have to concern himself with the actual physical location of that data, whether the data is in main memory, or on a hard disk.

In virtual memory systems, the term "physical memory" is used to define the physical address space seen by the operating system. Thus, the size of the physical address space will often be larger than the main memory available in the system. However, the data at any specific physical address must be accessed within the main memory. In operation of virtual memory systems, if requested data is not in main memory, then an operating system exception will be generated and the data will be transferred from secondary storage to main memory under the auspices of the operating system. If paging has been implemented, then a page validity bit may be associated with each page to indicate whether or not the page is in main memory. If the page validity bit indicates that a requested page is not in main memory, then the data is transferred to main memory and the page validity bit is set to indicate that the page is now in main memory.

Virtual memory systems can be categorized into two types: those with variable sized blocks, called "segments", and those with fixed size blocks, called "pages". One, or a combination of these two different methods are used to translate from a virtual address to a physical address. Using segmentation, a block of physical memory is allocated based on the amount of data to be stored. Thus, in a segmented memory structure, one segment may be large and the next may be very small. Segmentation uses memory efficiently, however there are disadvantages. Problems arise when a

2

segmented data structure is modified to be larger. The smaller block must be replaced, and a new, larger block must be found. This problem is substantial, particularly when it is recognized that data structures are often modified.

Paging is the other method of translating a virtual address to a physical address. Pages are fixed size blocks of memory that are mapped in specific locations in physical memory. A "page" is what the programmer sees (part of the logical or linear address), and the "page frame" is the physical memory itself. One or more tables are provided, each of which has a number of page table entries. Each page table entry specifies a specific page frame. The virtual address includes information that is indicative of the table and the page frame.

The use of pages provides advantages. Paging is generally efficient; there are no unused blocks but internal fragmentation (i.e., an unused portion of a page) can be a problem. Replacing a block is trivial. If data is modified to include additional information, then additional pages can be employed. From a design point of view, paging is generally preferred for bigger systems because paging makes allocation of memory easier.

The page size is an important architectural parameter. Choosing a page size is a question of balancing forces that favor a larger page size versus those favoring a smaller size. Advantages of larger page size include memory resources that are saved by use of larger page size, and efficient transfer of larger pages to and from secondary storage. Particularly, memory resources are saved because the size of the page table is inversely proportional to the page size, and thus a larger page size means a smaller table. Also, transferring larger pages is more efficient than transferring smaller pages. For example, the page size for the Intel 80386 and i486™ microprocessor is 4 Kbytes.

Most microprocessors employ a combination of segmentation and paging; specifically, each segment includes a number of pages. This approach to memory management is termed "paged segmentation". Such a system is disclosed in U.S. Pat. No. 4,972,338, entitled "Memory Management For Microprocessor System" issued to Crawford et al., which discloses a segmentation mechanism for translating a virtual memory address to a second memory address (linear address) that is applied to a two-level paging table to select a page frame. The Crawford et al. invention is embodied in the Intel 80386 microprocessor.

Briefly, as described in the Crawford et al. patent, segmentation translates a 48-bit virtual address to a 32-bit linear (intermediate) address. The virtual address includes a segment selector (14 bits) and segment offset (32 bits). Segment offset is the calculated result of the address calculation: scale index and the displacement. The segment selector comes from the segment register. The application programmer must specify a segment register and other components that give the offset. Most applications use only one segment.

FIG. 1 is an illustration of page translation in Intel 80386 microprocessor and the Intel i486™ microprocessor. A linear address including a page directory field, a page table field, and an offset field are applied to obtain an operand in a page frame. Within the computer memory, the data structures include page directories, page tables, and page frames. In operation, a page directory in memory is first chosen by a control register CR3. In the linear address, the page directory field selects a page directory entry that specifies a page table in physical memory. The page table field of the linear address selects a page table entry that specifies a page frame of data in physical memory. The offset field of the linear address is applied to select an operand within the page

5,802,605

3

frame. Thus, page translation for the Intel i486™ uses a tree structure with two levels of tables in memory.

Each of the tables occupies a 4 Kbyte block of physical memory and each entry is 4 bytes. Therefore each page directory address table has 1K 4-byte entries. The directory field of the linear address is a 10-bit index that selects one of the 1K entries in memory. Similarly, the page table has 1K 4-byte entries that are selected by the 10-bit page table field of the linear address. The 12-bit offset field of the linear address is applied directly to select any operand in the 4K bytes in the page frame. The page frame address is 20 bits in memory. With the 12 bit offset of the linear address, the translation provides a 32-bit physical address, which can access up to 4 Gbytes of physical memory.

Although large by past standards, a 4 Gbyte limit on physical memory is becoming a limitation, particularly for very large servers. For example, banking databases that store customer information may require tens of Gbytes. It would be an advantage to provide a memory management system that is compatible with 32-bit existing software for the 386 and i486™, and can access more than 4 Gbytes, for example up to 64 bits (1.8×10^{10} Gbytes). It would be a performance advantage if the memory access to this larger physical memory could be completed in the same, or a lesser amount of time. It would be another advantage if more than one page size could be selected to choose a page size corresponding to an efficient memory structure for a particular application program or type of data structure.

SUMMARY OF THE INVENTION

The present invention provides an address translator and a method for translating a linear address into a physical address for memory management in a computer. Different page sizes can be selected, and the address translator translates the selected page size. The address translator translates from a standard 32-bit linear address for compatibility with previous architectures. However, the translator can translate to a physical address that is larger than the linear address; i.e., greater than 32 bits (e.g., 36 bits). Translation occurs with no decrease in performance, i.e., no increase in access time. In alternative embodiments, the address translator can be expanded to accommodate greatly increased physical memory while retaining the ability to run pre-existing software. For example, the physical address of the preferred embodiment can be expanded to meet a new IEEE standard which requires a 64-bit physical address. Even with such a large physical address, compatibility is retained with systems that use a 32-bit physical address.

The apparatus of the present invention includes a memory management address translator for addressing a page frame of main memory using another computer supplied address. In the preferred embodiment, the translator translates a linear address supplied by a microprocessor in a computer that uses segmentation methods to obtain the linear address by translating a virtual address. However, it should be apparent to one skilled in the art that, in other computer architectures, the translator of the present invention could translate a computer supplied address obtained by another method.

The linear address that is applied to the translator includes an offset and a plurality of fields used to select entries in a plurality of tables. The tables include a directory pointer table that includes a group of directory pointers, a plurality of page table directories each of which includes a group of page directory entries, and a plurality of page tables each of which includes a group of page table entries. The size of the

4

entries in the page table directory is dependent upon the selected memory size, and is either 32-bits or 64-bits in the preferred embodiment. A first field of the linear address specifies a directory pointer in the directory pointer table, so that a directory pointer is selected that specifies a page directory table. A second field of the linear address specifies a page directory entry in the selected page directory table. A third field of the linear address specifies a page table entry so that a page frame in physical memory is selected.

The contents of the tables are stored in any suitable memory component such as main memory. The page table directories and the page tables are stored in main memory and accessed through a conventional memory access. In the preferred embodiment, the pointer table is stored in both main memory and in dedicated pointer table registers provided in the microprocessor. Preferably, the pointer table registers are invisible (i.e. not accessible) to a computer programmer, so that the registers cannot be written to or read directly by the programmer. A performance advantage is provided by the use of registers for the directory pointer table because a memory access is avoided.

In comparison to the memory management system of the i486™ microprocessor, the address translator described herein includes the directory pointer table in addition to the page table directories and the page tables. As a further feature, the directory pointer table is in main memory and also in dedicated on-chip registers. Register storage avoids one memory access for each translation, so that accessing the three tables to translate an address requires only two memory accesses at most, thereby saving computer time. Only one page size was supported on the i486™ microprocessor, but in the present invention, more than one page size is available. The size of the linear address remains 32 bits, however the table structure allows the physical address to be larger. For example, with extended addressing the physical address may be 36 bits (64 GB) and can be larger, 64 bits for example. The same size linear address permits present operating systems to take immediate advantage of extended physical addressing without change to their system software.

A larger page size can provide performance advantages in some instances, particularly for large amounts of contiguous data, by reducing the miss rate of the Translation Lookaside Buffer (TLB). With a larger page size, a single TLB entry can be used to access all the locations in the large page instead of using separate entries for accessing each small page. The single TLB entry for a large page saves time by avoiding the multiple TLB accesses that would occur if a small page size were used. Larger page sizes are useful, for example, in mapping memory-resident portions of the operating system, video display frame buffers, databases, and other large data structures. A large page size allows efficient reading and storing of these data structures.

In memory, a page is defined by a page frame that has boundaries set by its size. For example, a 4 Kbyte page has boundaries at each 4 Kbyte memory boundary, and a 2 Mbyte page has boundaries at each 2 Mbyte memory boundary. As discussed above, the address translator supports multiple page sizes; i.e. multiple page frame sizes. A single program may use multiple page sizes. The supported page frame sizes include: a first page frame size and a larger, second page frame size. The linear address includes a plurality of fields including selection fields and an offset field. The number and width of the selection fields is dependent upon the selected page frame size. For the first page frame size, the selection fields include a first selection field, a second selection field, and a third selection field. For

5,802,605

5

a second page frame size, the selection fields include only a first selection field. For a second page frame size with extended physical addressing, a second selection field is used in addition to the first selection field.

If the first page size is selected, then all three tables are used to translate a linear address to a physical address. However, if the second, larger page size is selected, then at least one of the tables is omitted. In the preferred embodiment with the normal addressing mode, the page tables and the pointer table are not used, and instead, the page directory entry points directly to the larger page frame. With extended physical addressing, the pointer tables and the page directory tables are used, but the page tables are not used and the page directory entry has a larger size and points directly to the larger page frame.

When the first page size is selected so that the linear address includes the first, the second, and the third selection field, the first selection field selects a directory pointer entry to select a directory table, the second selection field selects a selected page directory entry from the selected page directory table to select a page table, and the third selection field selects a page frame address from the selected page table. When the second page size and normal addressing is selected so that the linear address comprises only a first selection field and an offset field, the control register selects a page directory, and the first selection field points to a page frame. When the second page size and extended physical addressing is selected so that the linear address comprises a first field and a second field, the first selection field of the linear address selects a directory pointer to select a page directory table, and the second selection field selects a page frame from the selected page directory table.

In terms of a method, the present invention includes an address translation method for translating a linear address into a physical address. The translation method varies dependent upon page size and size of the physical address. Thus, two selections are made initially. A first selection selects either a small or a large page size. A second selection selects either normal addressing or extended physical addressing. The format of the linear address is selected to correspond to the selected page size although preferably the linear address size remains unchanged.

For the small page size in the described embodiment, the linear address has a pointer field, a directory field, a page field, and an offset field. The translation method for the small page size comprises the steps of applying the pointer field to select a directory pointer, applying the selected directory pointer to select a page directory, applying the page directory field to the selected page directory table to select a page table, applying the page table field to the selected page table to select a page frame address, and applying the page frame address and the offset field to provide a physical address.

If the second page size and normal addressing has been selected, then the second page size translation includes applying a control register to select a page directory, applying the first selection field in the linear address to the page directory to select a page frame, and applying the offset to select a particular operand. If the second page size and extended physical addressing has been selected, then the second page size translation method includes the steps of applying a directory pointer field to select a directory pointer, using the selected directory pointer to select a page directory, applying the page directory field to said selected page directory table to select a page frame address, and using the page frame address and the offset field to provide a physical address.

6

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagrammatic illustration of page translation for the i486™ microprocessor.

FIG. 2 is a diagrammatic illustration of page translation for the translator of the present invention when a small page frame size and extended addressing has been selected.

FIG. 3 is a diagrammatic illustration of page translation for the translator of the present invention when a large page frame size and normal addressing has been selected.

FIG. 4 is a diagrammatic illustration of page translation for the translator of the present invention when a large page frame size and extended physical addressing has been selected.

FIG. 5 is a diagram of a control register, a page directory entry, and a page table entry for a normal physical address space and a single page size that is the original and default mode for the Intel 80X86 family.

FIG. 6 is a diagram of a control register, a page directory entry, and a page table entry for a normal physical address space and a small page size.

FIG. 7 is a diagram of a control register and a page directory entry for a normal physical address space and a large page size.

FIG. 8 is a diagram of a control register, a page directory pointer, a page directory entry, and a page table entry for an extended physical address space and a small page size.

FIG. 9 is a diagram of a control register, a page directory pointer, and a page directory entry for an extended physical address space and a large page size.

FIG. 10 is a block diagram of a portion of the microprocessor and its connection to memory elements.

FIGS. 11 and 12 together show a flow chart of address translation for the microprocessor of the preferred embodiment, including selectable page size and memory size.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT OF THE INVENTION

The invention may be understood by reference to the following detailed description of the preferred embodiment and the figures wherein like parts are designated with like numerals throughout.

The preferred embodiment of the present invention includes a plurality of page tables at different levels whose structure and addressing is dependent upon the chosen page size and the number of physical address bits that are desired for the physical address. In the preferred embodiment, the tables are stored in digital form in memory, except where otherwise noted. The circuits necessary to perform the processing to be described can be implemented using a finite state machine, for example a programmable logic array (PLA), and other conventional logic.

Table 1, shown below, discloses features of the preferred embodiment. Specifically, Table 1 discloses selection of the page size and the size of the physical address. In the preferred embodiment, selection is controlled by two control bits in a control register (CR4). The two control bits are termed the "PAE" bit and the "PSE" bit.

5,802,605

7

TABLE 1

| Memory Management Features Controlled by PSE and PAE | | | |
|--|-----|--------------------|-----------------------|
| Control Bits | | Features Available | |
| PAE | PSE | Page Sizes | Physical Address Bits |
| 0 | 0 | 4KB | 32 |
| 0 | 1 | 4KB or 4MB | 32 |
| 1 | 0 | 4KB or 2MB | 36 |
| 1 | 1 | 4KB or 2MB | 36 |

The address mode is selected by the PAE bit. If PAE is "0" then normal 32-bit addressing is selected. If PAE is "1", then extended physical addressing is selected.

If PAE and PSE are both zero, then the page size is 4 KB and the physical address is 32-bits. This configuration is compatible with memory management in microprocessors including the Intel 80386 microprocessor and the 80486 microprocessor, both of which are commercially available and in wide use. Thus, software written for those microprocessors can be run with the memory management of the preferred embodiment, with PAE and PSE both set to zero.

If PAE is zero and PSE is set to one, then the size of the physical address is again 32-bits. However, the page size may be either 4 KB or 4 MB, depending upon a control bit in the page directory entry to be described below. The larger page size may be useful in applications where large blocks of memory are stored together, for example, video memory. In such an instance, the program sets the control bit when the page directory entry is entered into the directory.

If PAE is one and PSE is either set to one or reset to zero (i.e., "don't care"), then the physical address is extended to 36-bits, and the page size may be either 4 KB or 2 MB, depending upon a control bit in the page directory entry to be described. The 36-bit address provides extended addressing capabilities (64 Gbytes in the preferred embodiment) to address more memory than commercially available microprocessors including the Intel 80386 microprocessor and the 80486 microprocessor.

The above description and Table 1 show specific parameters applicable to the preferred embodiment of the invention. In other embodiments, the page size may be different, the number of control bits may be more or less, and the size of the physical address may be different. The smaller physical address size can be chosen to be compatible with prior art microprocessors, and the larger physical address size can be chosen for use with larger memory. In the preferred embodiment, the size of the physical address can be expanded by simply utilizing bits that are currently not utilized in the embodiment described herein. The physical address of 32-bits can address each memory block in a 4 GByte physical memory, while the extended physical address of 36-bits can address each memory block in a 64 GByte physical memory.

In the preferred embodiment, the physical address is translated from a linear address supplied by a microprocessor. In the preferred embodiment, the linear address is supplied by applying segmentation to a virtual address, however it will be apparent to one skilled in the art that the linear address may be supplied using other methods. The linear address is divided into fields used to translate the linear address into a physical address. Both the number of fields in the linear address, and the size of the fields may vary dependent upon the selected page size and physical address size, as will be described. In the preferred

8

embodiment, the number of fields and their size are both dependent upon the control parameters shown in Table 1, specifically, the PAE and PSE control bits, which specify the page size and the physical address size.

For page translation using a small 4K page size and a normal physical address space (4 GByte or less), the translation configuration is similar to that shown in FIG. 1, used in the Intel 80386 and 80486. That configuration is well known and described in many books, for example "Programming the 80386", by John H. Crawford and Patrick P. Gelsinger, Sybex, San Francisco, 1987.

Reference is made to FIG. 2 which is an illustration of page translation for a small page size and an extended physical memory. In the preferred embodiment, the small page size is 4 KB and the extended physical memory is 64 GByte or less. For the small page size and extended physical memory, three levels of tables are used. These three levels include a directory pointer table 20, a page table directory 22 and a page table 24. The directory pointer table 20 includes a plurality of pointer entries 30, the page table directory 22 includes a plurality of page directory entries 32, and the page table 24 includes a plurality of page table entries 34. A page frame 36 represents a block of data in memory having a size determined by the page directory entry 32, and by the PAE and PSE bits as described above with reference to Table 1. Each page frame 36 includes a plurality of operands 38 that are selected in a manner described below.

In order to access an operand, a control register 40 holds a value that points to the address of a particular directory pointer table 20 in memory. In the preferred embodiment, the directory pointer table 20 is actually stored in the microprocessor in dedicated registers, as will be described below. A linear address 41 for a small page size is divided into a pointer field 42, a directory field 44, a page field 46, and an offset field 48.

The pointer field 42 points to a particular pointer 30 in the directory pointer table 20 that is selected by the control register 40. In the preferred embodiment, the directory pointer table 20 includes four entries, and the pointer field 42 therefore includes 2-bits. The selected pointer 30 is applied to select a particular page table directory 22. Once the page table directory 22 is selected, a directory field 44 of the linear address selects a particular page directory entry 32. The selected page directory entry 32 points to one of the plurality of page tables 24. A page field 46 within the linear address selects a particular page table entry 34 in the selected page table 24. The selected page table entry 34 selects a page frame 36 in physical memory. The offset field 48 in the linear address points to an operand 38 in the specified page frame 36 in physical memory.

Reference is made to FIG. 3 which is an illustration of page translation for a larger (4 MB) page size with a normal size (4 GByte or less) physical memory. For translating the large page size, only one table level is used in place of the three levels of tables for the smaller size page as discussed previously with regard to FIG. 2. The control register 40 points directly to a page table directory 50. A directory field 52 in a linear address 53 points to a particular entry in the page directory. The format of the page directory entry will be described below. The selected page directory entry 54 points to a page frame 55 in physical memory. An offset field 56 in the linear address points to a particular operand 57 within the page frame 55.

Reference is made to FIG. 4 which is a configuration for pointing to a large page frame in physical memory. The embodiment of FIG. 4 is useful in the preferred embodiment

5,802,605

9

for extended addressing (64 GBytes or less). In other words, when a larger physical memory is provided, the embodiment of FIG. 4 provides additional information necessary to access the data within the larger physical memory. In the configuration of FIG. 4, two levels of tables are used, with the first level of tables being a directory pointer table 60. The control register 40 selects a directory pointer table 60 that includes a plurality of pointers 62. The format of the pointer will be described below. A pointer field 64 within a linear address 65 points to a particular pointer 62. The selected pointer points to a page table directory 66 resident within memory. A plurality of page directory entries 68 are included within the page table directory 66. The format of the page directory entries 68 will be described below. A directory field 70 within the linear address points to a particular page directory entry 68, which in turn selects a particular page frame 72 in physical memory. An offset field 74 in the linear address selects a particular operand 75 within the page frame 72. In the preferred embodiment, the size of the page frame 72 for extended addressing is 2 MB.

TABLE 2

| Memory Management Types and Modes Run | | | | | | | |
|---------------------------------------|---------|---------|--------|---------------|-------------------|-------------------|---------|
| Type | CR4 PAE | CR4 PSE | DIR PS | RE-LATED FIG. | PAGE SIZE (Bytes) | PHYSICAL ADDRESS | MODE |
| 0 | 0 | 0 | X | FIG. 5 | 4K | Normal (32-bit) | A (Def) |
| 1 | 0 | 1 | 0 | FIG. 6 | 4K | Normal (32-bit) | B |
| 2 | 0 | 1 | 1 | FIG. 7 | 4M | Normal (32-bit) | B |
| 3 | 1 | X | 0 | FIG. 8 | 4K | Extended (36-bit) | C |
| 4 | 1 | X | 1 | FIG. 9 | 2M | Extended (36-bit) | C |

For a more detailed description of the preferred embodiment, Table 2 sets forth memory management types and the modes that are run. Mode A is the default mode. This is the mode that is compatible with all other members of the X86 family of microprocessors, including the Intel 80386 and the Intel 80486 microprocessors. Mode B represents normal addressing (<4 GByte) and a capability for selecting one or two page sizes. Mode C represents extended addressing (a larger physical memory) and a capability for selecting one or two page sizes. Memory management for these modes has been discussed with reference to FIGS. 1-4. Each of these modes will be discussed further with regard to specific entries in the tables that are used in the preferred embodiment. These entries are specified as "Related Figures" in Table 2. It should be apparent to one skilled in the art that the specific arrangement of bits and the number of bits may vary between embodiments. In other embodiments, the actual page size for example may be different, or the physical address size may be different. FIGS. 5-9 are provided in order to comply with the full disclosure and provide an exemplary arrangement for implementing dual page size and extended physical addressing. Many of the bits and bit names have been used for Intel's i486™ micro processor, and are described in detail in the "i486™ Micro-programmer's Reference Manual", particularly at Section 5.3.3, available from Intel Corporation of Santa Clara, Calif.

Reference is made to FIG. 5 which illustrates the format of the registers and table entries pertinent to operation in Mode A. Mode A is the original and default type of the X86 family. It maps a standard 4K-byte page within a 32-bit physical address space. The look-up tables include both a page directory and a page table directory.

10

The page directory entry is 4 bytes wide and thus requires 30 bits to specify a unique address. The first 20 bits point to the page directory and will come from bits 31:12 of CR3. The offset within the page directory will come from the ten most significant bits of the linear address (31:22). One 4K-byte page can therefore fit 1K of 4-byte entries.

The details of operation in the default mode are well known with respect to Intel's 80386 and 80486 microprocessors and will not be repeated here. Reference is made to FIG. 1, which shows the table configuration that is implemented with the control register and table entries shown in FIG. 5.

Reference is now made to FIG. 6 which is a depiction of the format of the control register and the table entries. The look-up sequence for type 1 is virtually identical to that described for type 0 with two minor exceptions: the reserved (RSV) bits are checked for zero as soon as the P bit is validated, and the bit PDE.PS (Page Directory Entry Page Size) is zero for a small page size in type 1. (For a large page size, PDE.PS=1 which is type 2.) The table configuration for type 0 shown in FIG. 1 is also the table configuration for type 1.

Reference is made to FIG. 7 which is a depiction of the control register and the page directory entry for a large page size and a normal physical address space. The control register entries shown in FIG. 7 implement the table structure illustrated in FIG. 3. Type 2 results in a Translation Lookaside Buffer (TLB) entry that maps a 4 Mbyte page within a 32-bit physical address space. The look-up involves only a page directory entry as illustrated in FIG. 3.

The page directory entry (PDE) is retrieved and deemed valid in exactly the same way as the PDE of type 1. In this type of look-up the PDE.PS bit will be found set, thus no table look-up will take place. From this point on, the bit checking of the PDE will be virtually identical to the bit checking of the page table entry (PTE) of type 1. The only difference is that the U and W statuses are the PDE.U and PDE.W bits themselves.

Reference is made to FIG. 8 which illustrates the format for the control register and the table entries, particularly the page directory pointer format and the page directory entry format. Reference is also made to FIG. 2, which is implemented with the formats shown in FIG. 8.

The type 3 and extended addressing Mode C results in a TLB entry able to map a 4K-byte page within a 36-bit physical address space. The look-up uses a page directory pointer, a page directory entry, and a page table entry. There are four page directory pointers (PDPTR (3-0), located physically in registers within the P-unit. The page directory pointers are stored there by microcode when CR4.PAE is set, as part of a write to control register 3. The particular page directory pointer to look-up is selected by bits 31:30 of the linear address. The P bit is checked, to see that it points to a valid page directory, and then the RSV bits of the pointer are verified to be zero. If the RSV bits are not all zero a page fault is returned.

Within this Mode C, the PDE is eight bytes wide and thus requires 33 bits to specify a unique address. The first 24 bits point to the page directory and come from the selected page directory pointer bits 35:12. The particular page directory entry is selected from the linear address bits 29:21. Within one 4K-byte page, 512 eight byte entries can fit. The bit checks of the PDE are handled in the same fashion as those of the PDE for type 1, with the main difference being that there are 28 more RSV bits.

The page table entry (PTE) is also eight bytes wide and accessed similarly to the PDE. The only difference is that the

5,802,605

11

page table is at bits 35:12 of the PDE in the offset is in the linear address bits 20:12. Similarly, the bit checks of the PTE are handled similarly as the PTE of type 1, with the main difference being that there are 28 more RSV bits. The TLB entry for Mode C occupies the full Translation Lookaside Buffer of 36 bits.

Reference is made to FIG. 9 which depicts the format of the control register and the entries into the page directories and the page directory pointers. These register formats implement the embodiment of FIG. 4, for the extended physical addressing and a large page size. Specifically, type 4 and mode C results in a TLB entry able to map a 2M-byte page within a 36-bit physical address space. The look-up involves selection of a pointer from the page directory pointers, which select a particular page directory as illustrated in FIG. 4.

The PDE is retrieved and deemed valid in exactly the same way as the PDE of type 3. In this type of look-up the PDE.PS bit will be found set, thus no table look-up will take place. From this point on the bit checking of the PDE will be virtually identical to the bit checking of the PTE of type 3. The main difference is that the U and W statuses are the PDE.U and the PDE.W bits themselves.

In order to implement extended physical addressing, it should be apparent that the buses and particularly the TLB and the TLB bus be able to support the size of the extended physical address. In the preferred embodiment, the size of the physical address is 36 bits, and therefore the TLB and the TLB bus are implemented to store and communicate this size physical address.

Reference is made to FIG. 10, which is an illustration of the relationship between a microprocessor 100 and a computer memory 102. The microprocessor includes a control unit 104 which has numerous registers provided therein including a CR3 register 106 and a CR4 register 108. The microprocessor 100 also includes a page unit 110 which includes logic and additional circuitry including hardware, PLA programming sequencing control, and pointer registers 112. The page unit 110, sometimes referred to as the "P-unit," controls the address translation as described above. For example, the P-unit controls all table manipulations and directs transfers with main memory for missing TLB entries.

The computer memory 102 includes a main memory 114 that may, for example, be a Random Access Memory (RAM) that is easily accessible by a user. The computer memory 102 also may include one or more secondary memories such as a disk memory 116. A secondary memory typically requires more time for each memory access than the main memory 114. As described above, the page directories and the page tables are located in computer memory 102. The directory pointer table is also located in computer memory 102, but additionally its same entries are located within the page unit 110, specifically within the pointer registers 112. Loading the pointer registers 112 from the computer memory 102 occurs in the preferred embodiment whenever the CR3 register 106 is loaded. Because the pointers are in the registers 112, within the page unit 110 access time is substantially reduced and therefore performance is increased.

The term "physical memory" defines the physical address space seen by the operating system. The size of the physical address space is usually larger than the available main memory, but data at a specific physical address must be accessed within the main memory. A page validity bit is associated with each page table entry to indicate whether or not the page is in main memory. If the page validity bit

12

indicates that a requested page is not in main memory, then an operating system exception is generated and the data is transferred from secondary storage to main memory. Then, the page validity bit is set to indicate that the page is now in main memory.

Reference is made to the flow chart of FIGS. 11 and 12, which together illustrate operation of page translation in the preferred embodiment. The flow chart of FIGS. 11 and 12 shows translation for a selectable memory size, and a selectable page size. Following initialization 120 at the top of the chart, a memory size is selected in a decision box 121. If the memory size is selected to be "small", then the table entries are formatted accordingly and set up with a size of 32 bits, as illustrated in an operation box 122. However, if a "large" memory size is selected, then the pointers and the table entries are set up to a 64-bit size, as illustrated in operation box 124. From both operation boxes 122 and 124 control flows to decision box 126. At decision box 126, a wait loop is performed until a new task is created. Whenever a new task is created, the number of pages and page sizes for each table are selected in operation box 128. Following operation box 128, each table is set up in operation box 129. Table set up includes setting the page bit based upon the selected page size.

Following the above operations, the page size is selected as illustrated in a decision box 130. If a "small" size is selected, then the page bit remains clear in the page directory entry as illustrated in a box 131. However, if a "large" page size is selected, then the page directory entries that are created have their page bit set, as illustrated in operation box 132. Upon completion of table set up in operation box 129, it is determined if all tables have been set up at decision box 134. If all the tables have not been set up for the new task, then boxes 130-132 are repeated for each table until all tables have been set up for the task.

After all the tables for the new task have been set up, a wait loop is entered at decision box 136 until the task is selected. When the task is selected, it is determined whether or not the PAE bit is set, in decision box 138. If the PAE bit is not set, then the processor is ready to perform memory reference. If the PAE bit is set, then the pointer registers are loaded into the page unit 110, in operation box 140 and the processor is ready to perform memory reference. Because the set up is now complete, the processor stands ready to request a memory reference as illustrated in a decision box 142. Moving the reference from FIG. 11 to FIG. 12 after the memory reference has been requested, the selected size of memory is reviewed as illustrated in a decision box 142. If a small size has been previously selected for the box 142; then the contents of the control register are applied to select a particular page directory, as illustrated in the operation box 144. However, if a large memory size is selected, then, as illustrated in a operation box 146, a pointer is selected by applying a pointer field of the linear address. Subsequently, as illustrated in a box 148, a page directory is selected by applying the selected pointer. After the page directory has been selected then, as illustrated in a box 150 the directory field of the linear address is applied to select a directory entry. Next, as illustrated in a decision box 152, the directory entry is checked to see whether or not the page bit has been set for a large page size. If a large page size has been selected, then the page frame is selected directly by applying the directory entry, as illustrated in a box 154. However if the page bit was not set for a large page size (i.e., a small page size has been selected) then, as illustrated in a box 156 a page table is selected using the directory entry. Next, as illustrated in a box 158, the page field of the linear address

5,802,605

13

is applied to the page table to select a page frame. After a page frame has been selected, either in the box 158 or the box 154, then as illustrated in the box 160 the offset field of the linear address is applied to select an operand in the page frame.

The invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiment is to be considered in all respects only as illustrative and not restrictive and the scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing descriptions. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed is:

1. A processor generating linear addresses having no more than N bits, said processor comprising:

a control unit having stored therein one or more control bits; and

a paging unit coupled to said control unit to receive said one or more control bits, said paging unit supporting translation of said linear addresses into physical addresses in a first physical address space having no more than 2^N locations that can be addressed while said one or more control bits are in a first state, said paging unit supporting translation of said linear addresses into physical addresses in a second physical address space having more than 2^N locations that can be addressed while said one or more control bits are in a second state.

2. The processor of claim 1, wherein said paging unit supports a first and second page frame size.

3. The processor of claim 2, wherein said first page frame size is 4K and said second page frame size is 2M or 4M.

4. An address translator for physical memory, said address translator translating a linear address having no more than N bits into a physical address, said address translator comprising:

memory size selection means for selecting a physical address size to be a first address size having no more than 2^N locations that can be addressed or a second address size having greater than 2^N locations that can be addressed;

one or more page directories, each having a group of page directory entries, said linear address having a page directory field for selecting a page directory entry;

a plurality of page tables, each having a group of page table entries, said linear address having a format, said format selected from a plurality of predetermined formats, at least one of said plurality of predetermined formats having a page table field for selecting a page table entry; and

means, responsive to said memory size selection means, for formatting the page table entries to a first page table entry size if the first address size has been selected or to a second page table entry size if the second address size has been selected, said second page table entry size being larger than the first page table entry size.

5. The address translator of claim 4, further comprising means, responsive to said memory size selection means, for formatting the directory entries to a first directory entry size if the first address size has been selected or to a second directory entry size if the second address size has been selected, said second directory entry size being larger than the first directory entry size.

6. The address translator of claim 4 further comprising a directory pointer table including a group of directory pointer table entries if said second address size has been selected,

14

said format of said linear address having a pointer selection field for selecting one of said directory pointer table entries if said second address size has been selected.

7. The address translator of claim 6, further comprising a plurality of registers for holding said directory pointer table entries.

8. The address translator of claim 7, wherein the registers are dedicated for holding said directory pointer table entries.

9. The address translator of claim 6 further comprising a control register containing a value, said value for selecting a page directory from said one or more page directories if the first address size has been selected, said value for selecting said directory pointer table if the second address size has been selected.

10. The address translator of claim 4 further comprising page size selection means for selecting a page size to be a first page size or a second page size, said second page size being larger than the first, said page size selection means including a flag in each page directory entry that is indicative of the selected size of one or more corresponding pages.

11. A method for use by a processor to translate a linear address having a size of no more than N bits into a physical address, said method using a plurality of tables including one or more page directories with page directory entries and at least one page table with page table entries, said method comprising the computer implemented steps of:

said processor altering a physical address mode indicator, said physical address mode indicator identifying said physical address size to be a first address size or a second address size, the first address size having no more than 2^N locations that can be addressed, the second address size having greater than 2^N locations that can be addressed;

if the first address size has been selected, then said processor setting the page directory entries and the page table entries to a first entry size;

if the second address size has been selected, then said processor setting the page directory entries and the page table entries to a second entry size that is larger than the first entry size; and

translating said linear address into said physical address using those of said plurality of tables having a corresponding field in said linear address.

12. The address translation method of claim 11, further comprising the steps of:

providing a flag in each page directory entry; and

for each of said page directory entries, performing the steps of selecting a page size to be said first page size or said second page size, said second page size being larger than the first page size, and

altering said flag to indicate said page size.

13. The address translation method of claim 12 further comprising the steps of:

if the first address size has been selected, then applying the contents of a control register to select one of said one or more page directories; and

if the second address size has been selected, then applying a pointer field of the linear address to select a pointer from a directory pointer table, and applying said selected pointer to select one of said one or more page directories.

14. The address translation method of claim 13, further comprising the steps of:

applying a directory field from the linear address to select one of said page directory entries as a selected page directory entry;

5,802,605

15

testing said flag in the selected page directory entry;
 if said flag in said selected page directory indicates the
 second page size has been selected, then applying the
 page directory entry to select a page frame;
 if said flag in said selected page directory indicates the
 first page size has been selected, then applying the page
 directory entry to select a page table, and applying a
 page table field from the linear address to select a page
 frame from the page table; and
 applying an offset field from the linear address to the
 selected page frame to select an operand.

15. An address translation method for use by a processor
 to translate a linear address having no more than N bits into
 a physical address, said linear address having a first field and
 a second field, said translation method comprising the com-
 puter implemented steps of:

- (a) executing one or more instructions on said processor
 to store a physical address mode indicator, said physi-
 cal address mode indicator identifying which of a first
 physical address size and a second physical address
 size will be utilized by said processor, said first physical
 address size having no more than 2^N locations that can
 be addressed, said second physical address size having
 more than 2^N locations that can be addressed;
- (b) said processor formatting page directory entries in one
 or more page directories and page table entries in a
 plurality of page tables to be compatible with the
 physical address size identified by said physical address
 mode indicator;
- (c) if said first physical address size is identified by said
 physical address mode indicator, applying both a value
 and said first field to select a page directory entry as a
 selected page directory entry, wherein said value is
 contained in said processor and said first field is applied
 as a directory field;
- (d) if said second physical address size is identified by
 said physical address mode indicator, performing the
 steps of
 - (d)(1) applying both said value and a first portion of
 said first field to select one of a plurality of directory
 pointer table entries as a selected directory pointer
 table entry, wherein said first portion of said first field
 is applied as a pointer field;
 - (d)(2) selecting one of a plurality of page directories as
 said selected page directory using said selected
 directory pointer table entry, and
 - (d)(3) applying a second portion of said first field as a
 directory field to select a page directory entry as a
 selected page directory entry from said selected page
 directory; and
- (e) using said selected page directory entry and said
 second field to provide a physical address, applying a
 first portion of said second field and a second portion of
 said second field if said selected page directory entry
 indicates said physical address is located in a page
 frame having a first page frame size, and applying said
 second field as an offset field if said selected page
 directory entry indicates said physical address is
 located in a page frame having a second page frame
 size, wherein said first portion of said second field is
 applied as a page table field and said second portion of
 said second field is applied as an offset field.

16. An address translation apparatus for use by a proces-
 sor to translate a linear address having no more than N bits
 into a physical address, said linear address having a first field
 and a second field, said address translation apparatus com-
 prising:

16

means for executing one or more instructions on said
 processor to store a physical address mode indicator,
 said physical address mode indicator identifying which
 of a first physical address size and a second physical
 address size will be utilized by said processor, said first
 physical address size having no more than 2^N locations
 that can be addressed, said second physical address size
 having more than 2^N locations that can be addressed;
 means for formatting page directory entries in one or
 more page directories and page table entries in a
 plurality of page tables to be compatible with the
 physical address size identified by said physical address
 mode indicator;

means for applying both said first field and a value to
 select a page directory entry as a selected page direc-
 tory entry if said first physical address size is identified
 by said physical address mode indicator, wherein said
 first field is applied as a directory field and said value
 is stored in said processor;

means for applying both a first portion of said first field
 and said value to select one of a plurality of directory
 pointer table entries as a selected directory pointer table
 entry, selecting one of a plurality of page directories as
 said selected page directory using said selected direc-
 tory pointer table entry, and applying a second portion
 of said first field as a directory field to select a page
 directory entry as a selected page directory entry from
 said selected page directory if said second physical
 address size is identified by said physical address mode
 indicator, wherein said first portion of said first field is
 applied as a pointer field; and

means for applying said selected page directory entry and
 said second field to provide a physical address.

17. The address translation apparatus of claim 16, wherein
 said means for applying said selected page directory entry
 and said second field to provide a physical address further
 includes applying a first portion of said second field and a
 second portion of said second field if said selected page
 directory entry indicates said physical address is located in
 a page frame having a first page frame size, and if said
 selected page directory entry indicates said physical address
 is located in a page frame having a second page frame size,
 said second field is applied as an offset field, wherein said
 first portion of said second field is applied as a page table
 field and said second portion of said second field is applied
 as an offset field.

18. An address translation method for translating a linear
 address into a physical address, said address translation
 method including a first address translation method for a first
 page size and a second translation method for a second page
 size larger than the first page size, said address translation
 method comprising the computer implemented steps of:

- (a) determining which of a plurality of states is indicated
 by one or more control bits, wherein a first of said
 plurality of states indicates at least said first page size
 and said second page size can be simultaneously
 supported, wherein a second of said plurality of states
 indicates only one page frame size can be supported;
- (b) if said one or more control bits are in said first of said
 plurality of states, then selecting either said first page
 size or said second page size, and if said one or more
 control bits are in said second of said plurality of states,
 then selecting said first page size;
- (c) if the first page size has been selected, then performing
 the steps of
 - (c)(1) applying a directory pointer field to select one of
 a plurality of directory pointers,

5,802,605

17

- (c)(2) using said selected directory pointer to select one of a plurality of page directories,
- (c)(3) applying the page directory field to said selected page directory table to select one of a plurality of page tables, 5
- (c)(4) applying the page table field to said selected page table to select one of a plurality of page frame addresses, and,
- (c)(5) using said page frame address and the offset field to provide a physical address; and 10
- (d) if the second page size has been selected, then performing the steps of

18

- (d)(1) applying a directory pointer field to select one of a plurality of directory pointers,
- (d)(2) using said selected directory pointer to select one of a plurality of page directories,
- (d)(3) applying the page directory field to said selected page directory table to select one of a plurality of page frame addresses, and
- (d)(4) using said page frame address and the offset field to provide a physical address.

* * * * *

EXHIBIT 4



US005819101A

United States Patent

[19]

[11] Patent Number: 5,819,101

Peleg et al.

[45] Date of Patent: Oct. 6, 1998

[54] METHOD FOR PACKING A PLURALITY OF PACKED DATA ELEMENTS IN RESPONSE TO A PACK INSTRUCTION

[75] Inventors: Alexander Peleg; Yaakov Yaari, both of Haifa, Israel; Millind Mittal, South San Francisco; Larry M. Mennemeier, Boulder Creek, both of Calif.; Benny Eitan, Haifa, Israel

[73] Assignee: Intel Corporation, Santa Clara, Calif.

[21] Appl. No.: 897,283

[22] Filed: Jul. 21, 1997

Related U.S. Application Data

[62] Division of Ser. No. 799,468, Feb. 13, 1997, which is a continuation of Ser. No. 626,698, Apr. 2, 1996, abandoned, which is a continuation of Ser. No. 349,047, Dec. 2, 1994, abandoned.

[51] Int. Cl.⁶ G06F 9/30

[52] U.S. Cl. 395/800.22; 395/562; 395/564

[58] Field of Search 395/562, 564, 395/800.22

[56] References Cited

U.S. PATENT DOCUMENTS

| | | | |
|-----------|---------|--------------------|------------|
| 3,711,692 | 1/1973 | Batcher | 364/715.09 |
| 3,723,715 | 3/1973 | Chen et al. | 364/786.04 |
| 4,161,784 | 7/1979 | Cushing et al. | 364/748.18 |
| 4,393,468 | 7/1983 | New | 364/749 |
| 4,418,383 | 11/1983 | Doyle et al. | 395/307 |
| 4,498,177 | 2/1985 | Larson | 371/52 |
| 4,707,800 | 11/1987 | Montrone et al. | 364/788 |
| 4,771,379 | 9/1988 | Ando et al. | 395/800.42 |
| 4,989,168 | 1/1991 | Kuroda et al. | 364/715.09 |
| 5,095,457 | 3/1992 | Jeong | 364/758 |
| 5,187,679 | 2/1993 | Vassiliadis et al. | 364/786.01 |
| 5,423,010 | 6/1995 | Mizukami | 341/60 |
| 5,594,437 | 1/1997 | O'Malley | 341/67 |
| 5,625,374 | 4/1997 | Turkowski | 345/113 |
| 5,680,161 | 10/1997 | Lehman et al. | |

OTHER PUBLICATIONS

J. Shipnes, *Graphics Processing with the 88110 RISC Microprocessor*, IEEE (1992), pp. 169–174.

Errata to MC88110 Second Generation RISC Microprocessor User's Manual, Motorola, Inc. (1992), pp. 1–11.

MC88110 Programmer's Reference Guide, Motorola, Inc. (1992), pp. 1–4.

i860™ Microprocessor Family Programmer's Reference Manual, Intel Corporation (1992), Ch. 1, 3, 8, 12.

R.B. Lee, *Accelerating Multimedia With Enhanced Microprocessors*, IEEE Micro (Apr. 1995), pp. 22–32.

MC88110 Second Generation RISC Microprocessor User's Manual, Motorola, Inc. (1991).

TMS320C2x User's Guide, Texas Instruments (1993) pp. 3–2 through 3–11; 3–28 through 3–34; 4–1 through 4–22; 4–41; 4–103; 4–199 through 4–120; 4–122; 4–150 through 4–151.

L. Gwennap, *New PA-RISC Processor Decodes MPEG Video*, Microprocessor Report (Jan. 1994), pp. 16, 17.

(List continued on next page.)

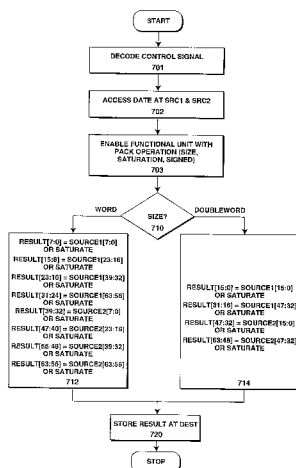
Primary Examiner—Kenneth S. Kim

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] ABSTRACT

A method for manipulating packed data in a computer system. The method includes the steps of decoding a Single Instruction Multiple Data (SIMD) pack instruction. The instruction identifies a first and second packed data respectively including a first plurality of data elements and a second plurality of data elements. Each data element consists of a separate multiple bit data field, wherein each data element in the first plurality of data elements corresponds to a data element in the second plurality of data elements in a respective position. The method further includes the step of simultaneously copying, in response to the pack instruction, a part of each data element in the first plurality of data elements and a part of each corresponding data element in the second plurality of data elements into a third packed data as a plurality of separate result data elements.

16 Claims, 17 Drawing Sheets



5,819,101

Page 2

OTHER PUBLICATIONS

SPARC Technology Buisness, *Ultra SPARC Multimedia Capabilities On-Chip Support for Real-Time Video and Advanced Graphics*, Sun Microsystems (Sep. 1994).

Y. Kawakami et al., *LSI Applications: A Single-Chip Digital Signal Processor for Voiceband Applications*, Solid State Circuits Conference, Digest of Technical Papers; IEEE International (1980).

B. Case, *Philips Hopes to Displace DSPs with VLIW*, Microprocessor Report (Dec. 94), pp. 12-15.

N. Margulis, *i860 Microprocessor Architecture*, McGraw Hill, Inc. (1990) Ch. 6, 7, 8, 10, 11.

Pentium Processor User's Manual vol. 3: Architecture and Programming Manual, Intel Corporation (1993), Ch. 1, 3, 4, 6, 8, and 18.

MC88110 Second Generation-RISC Microprocessor User's Manual pp. 1-23 (Sep. 1992), pp. 2-1 through 2-22, 3-1 through 3-32, pp. 5-1 through 5-25, pp. 10-62 through 10-71, Index 1 through 17.

INTEL i750, i860™, i960 Processors and Related Products pp. 1-3 (1993).

U.S. Patent

Oct. 6, 1998

Sheet 1 of 17

5,819,101

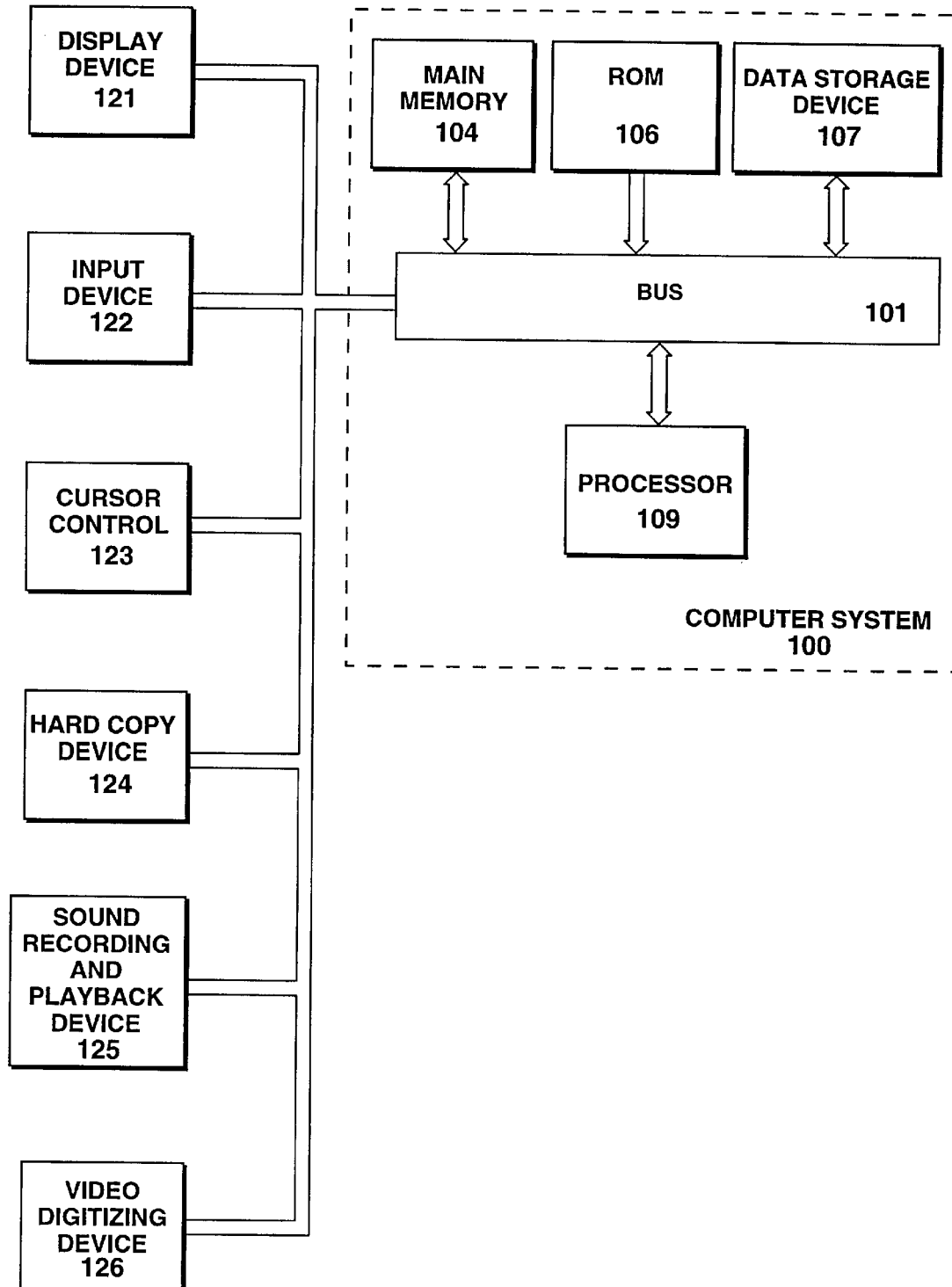


Figure 1

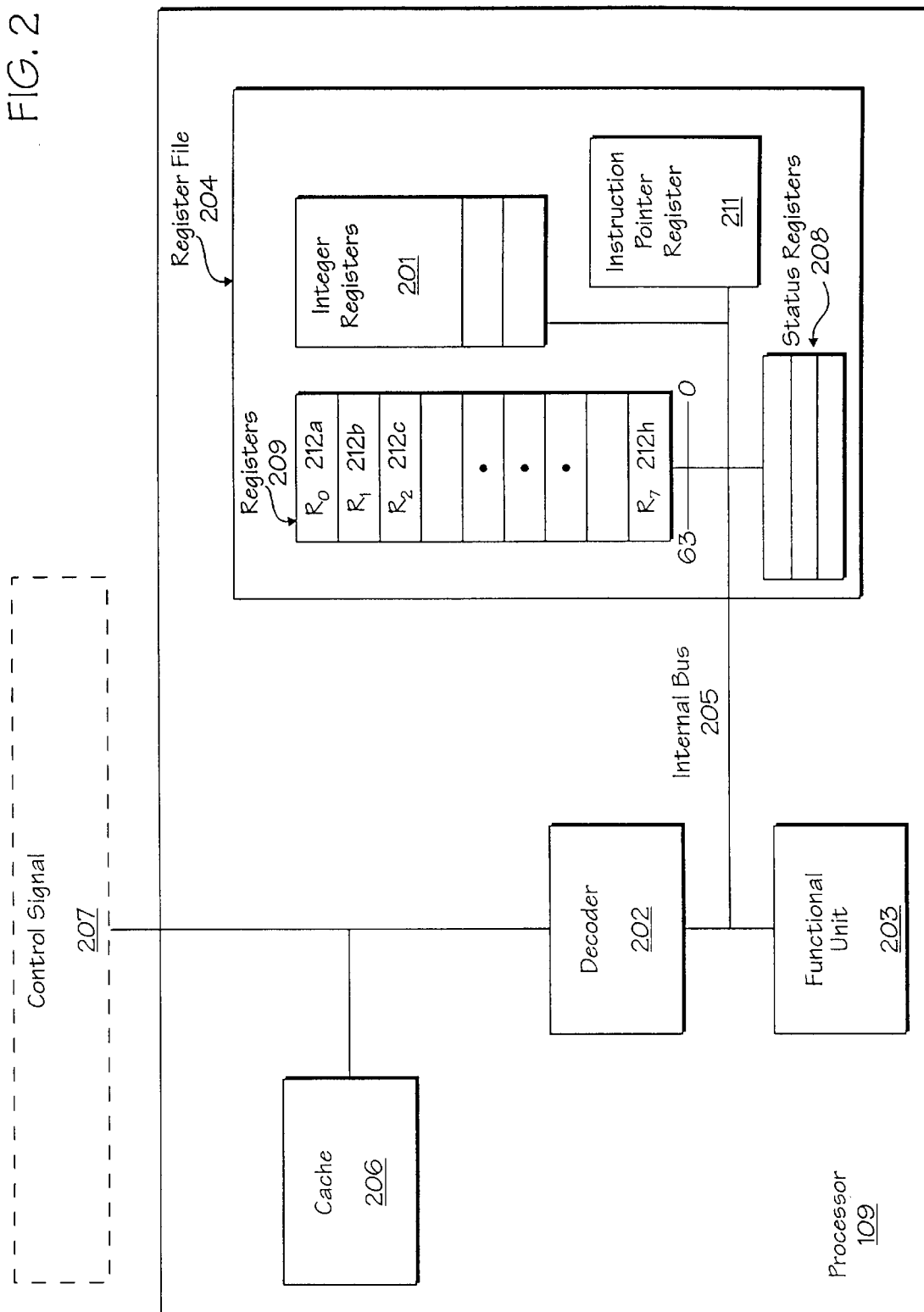
U.S. Patent

Oct. 6, 1998

Sheet 2 of 17

5,819,101

FIG. 2



U.S. Patent

Oct. 6, 1998

Sheet 3 of 17

5,819,101

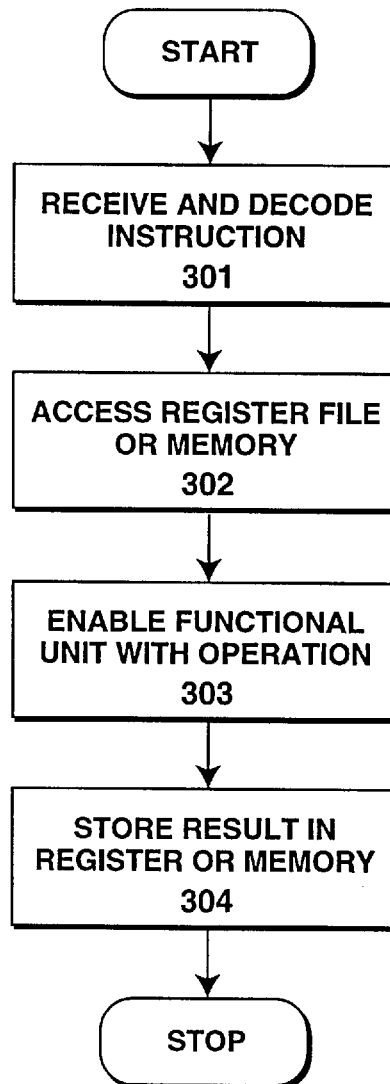


Figure 3

U.S. Patent

Oct. 6, 1998

Sheet 4 of 17

5,819,101

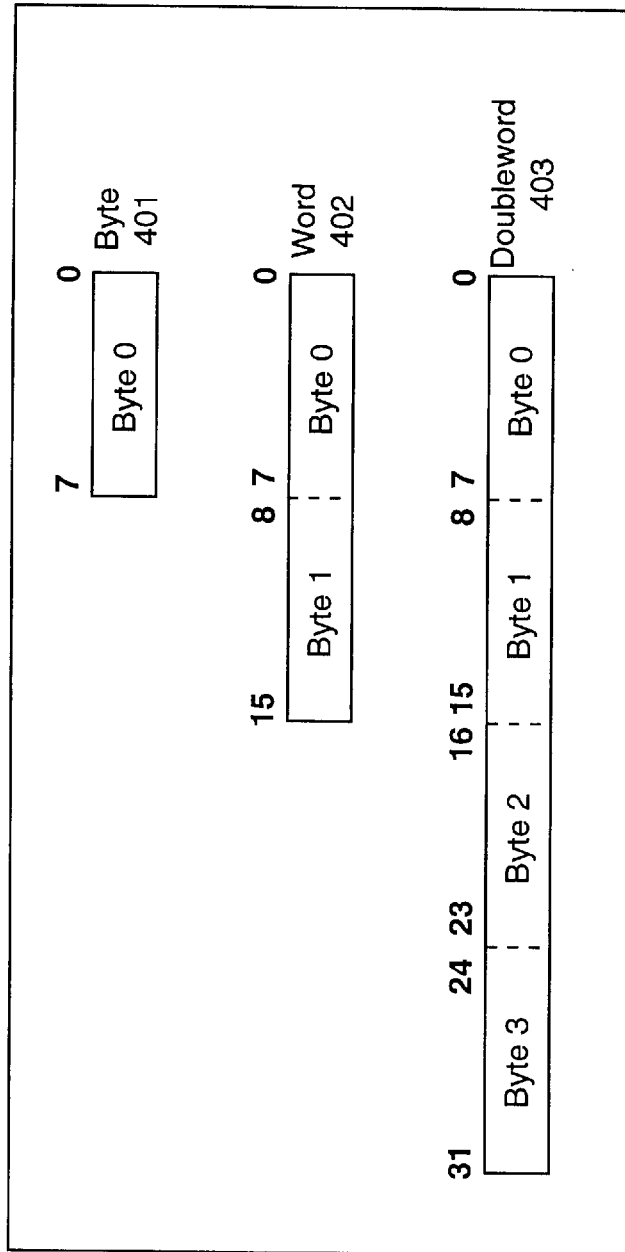


Figure 4a

U.S. Patent

Oct. 6, 1998

Sheet 5 of 17

5,819,101

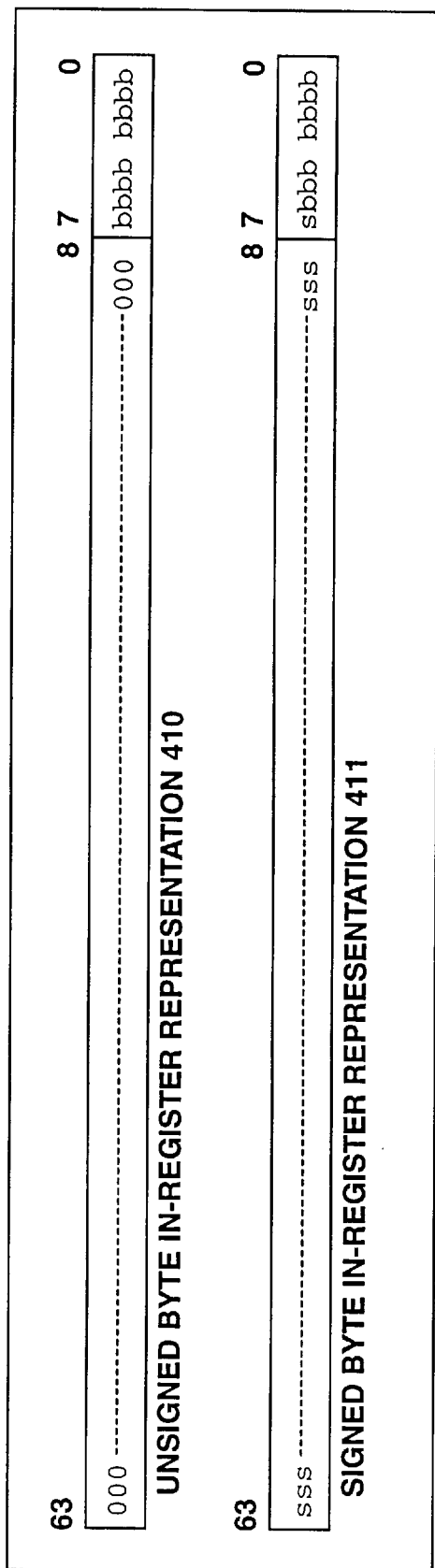


Figure 4b

U.S. Patent

Oct. 6, 1998

Sheet 6 of 17

5,819,101

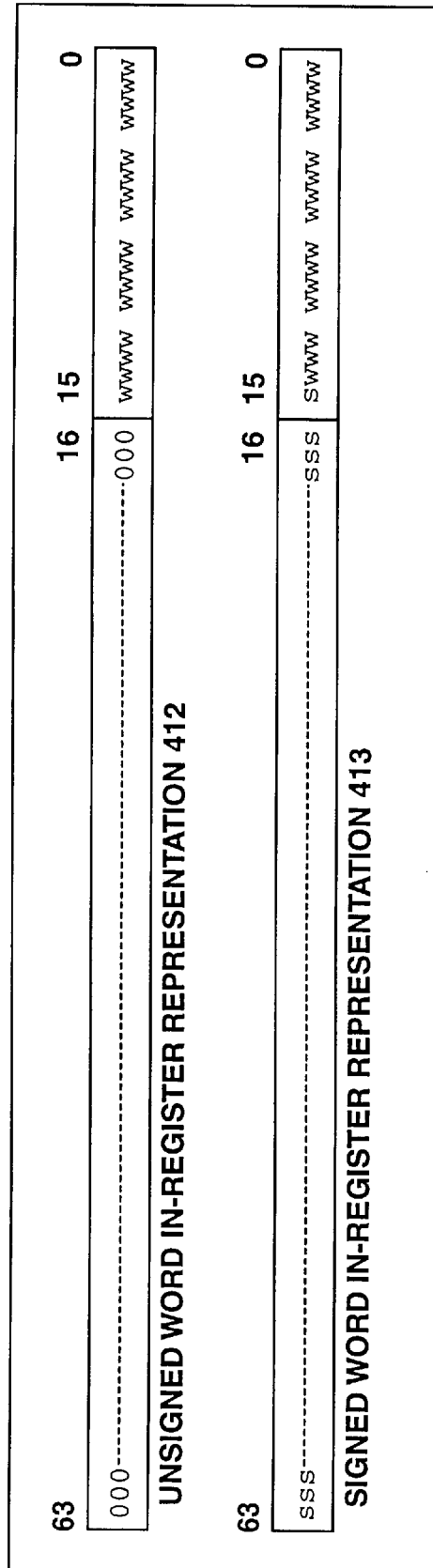


Figure 4c

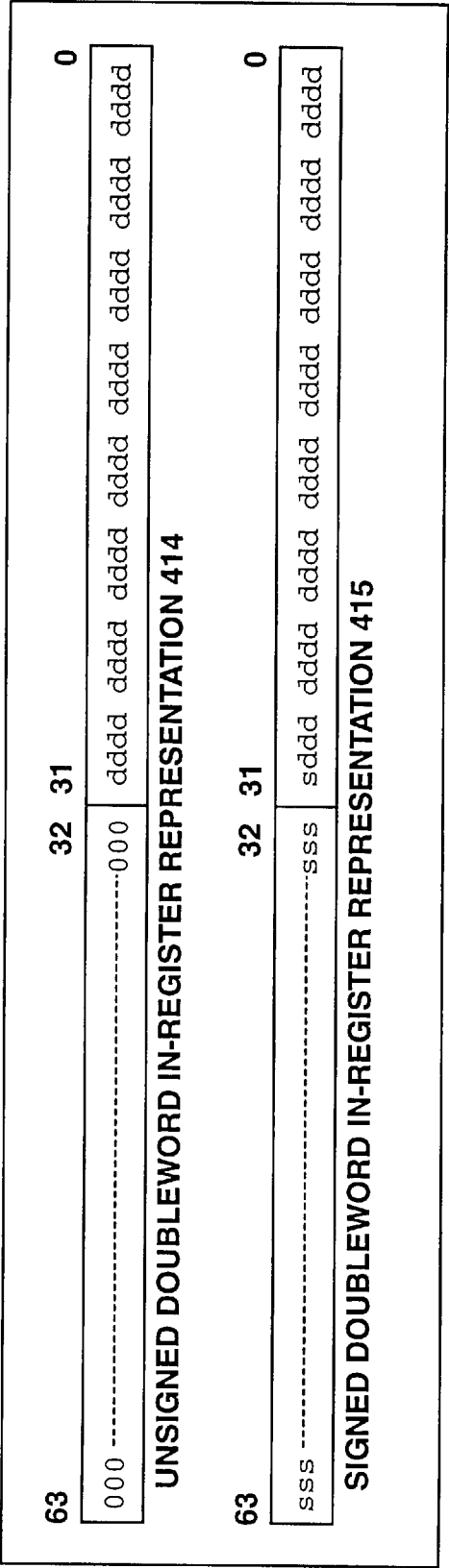


Figure 4d

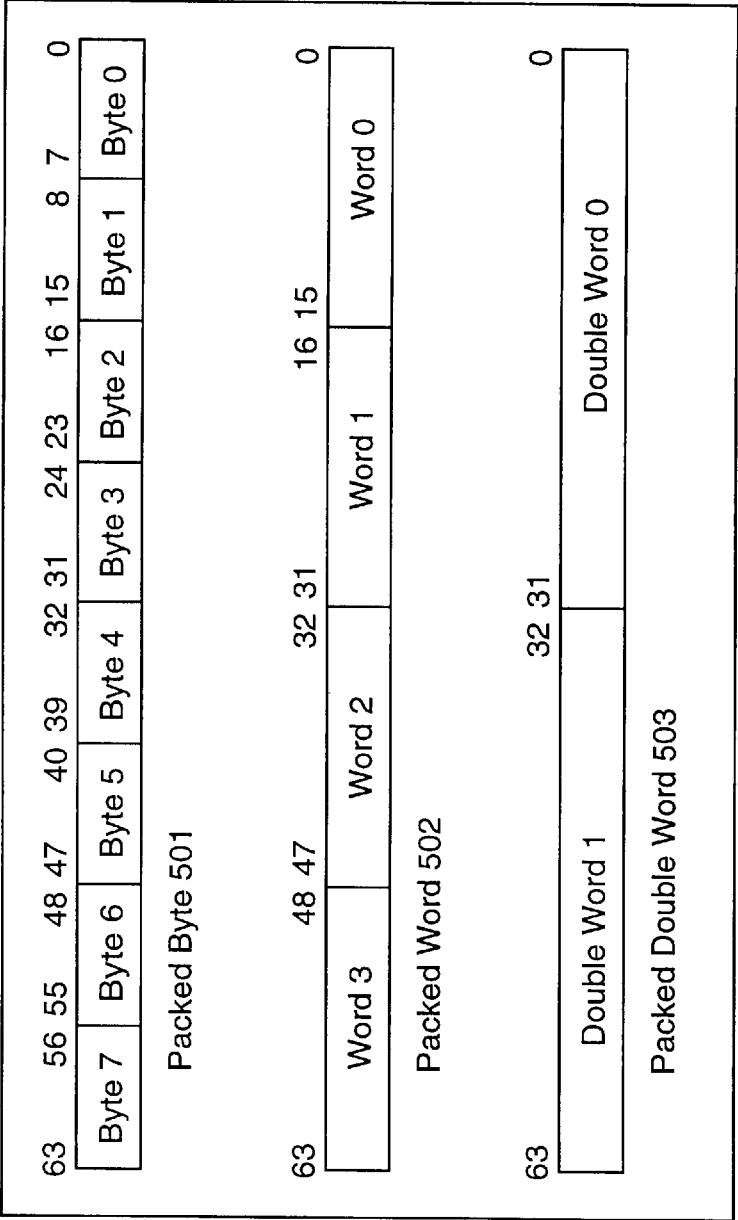


Figure 5a

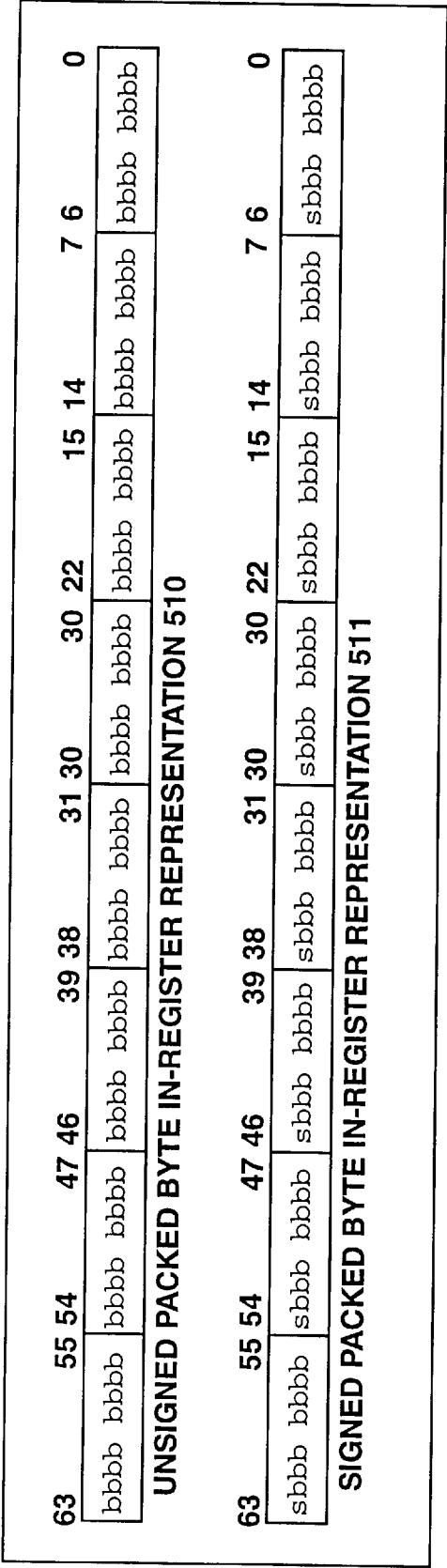


Figure 5b

U.S. Patent

Oct. 6, 1998

Sheet 10 of 17

5,819,101

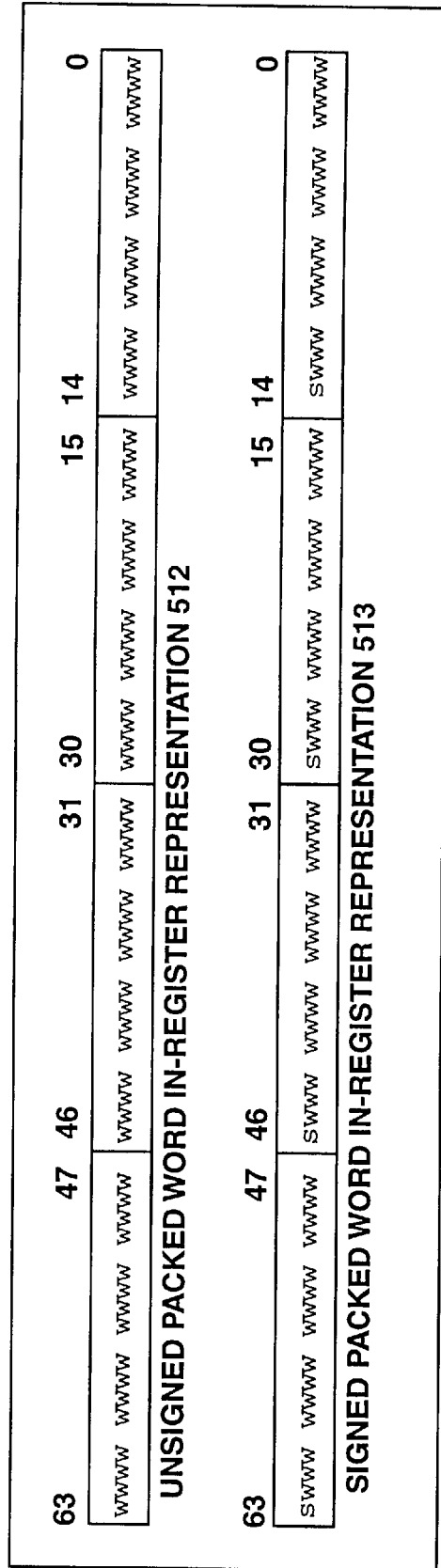
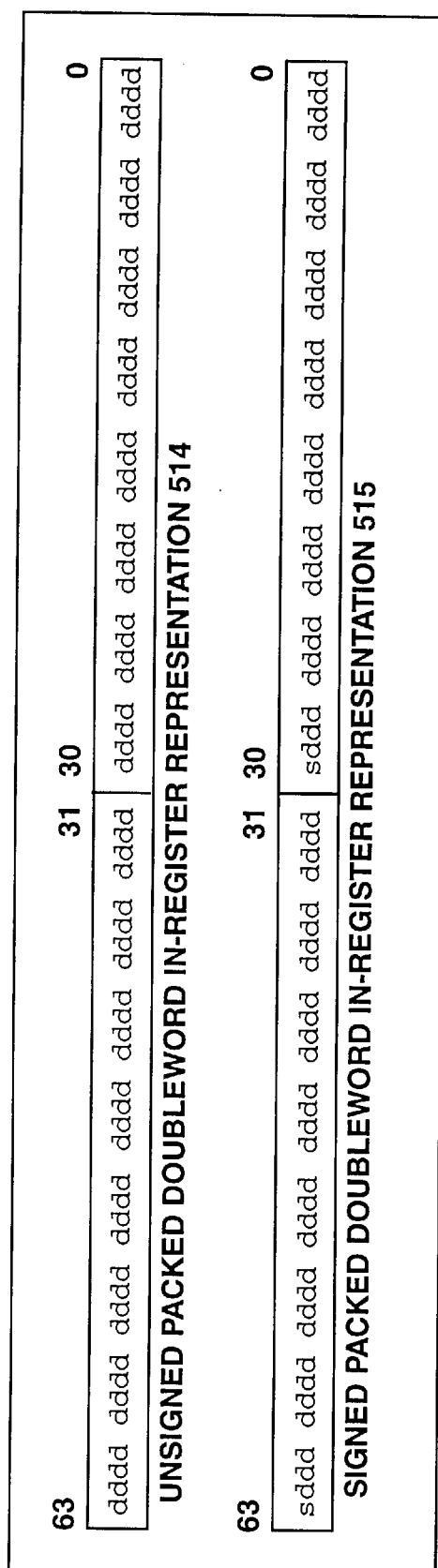


Figure 5c



U.S. Patent

Oct. 6, 1998

Sheet 12 of 17

5,819,101

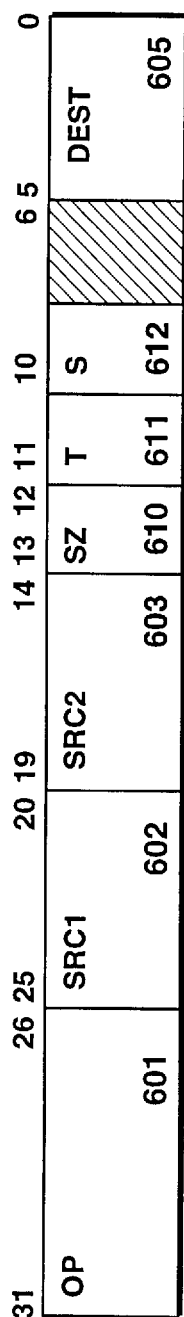


Figure 6a

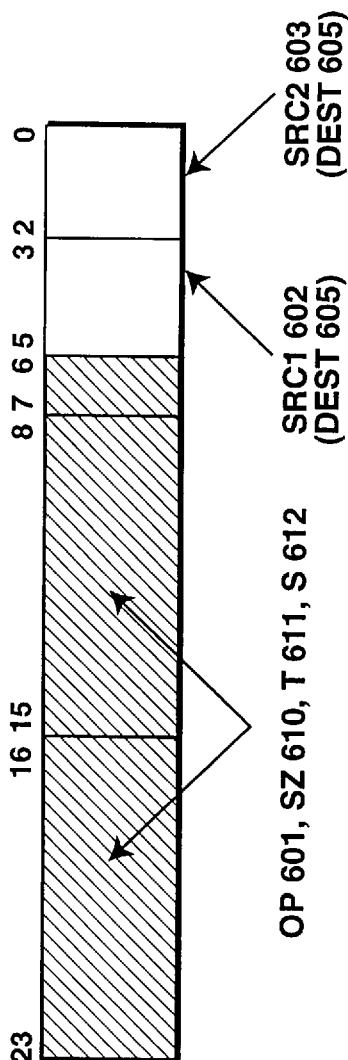


Figure 6b

U.S. Patent

Oct. 6, 1998

Sheet 13 of 17

5,819,101

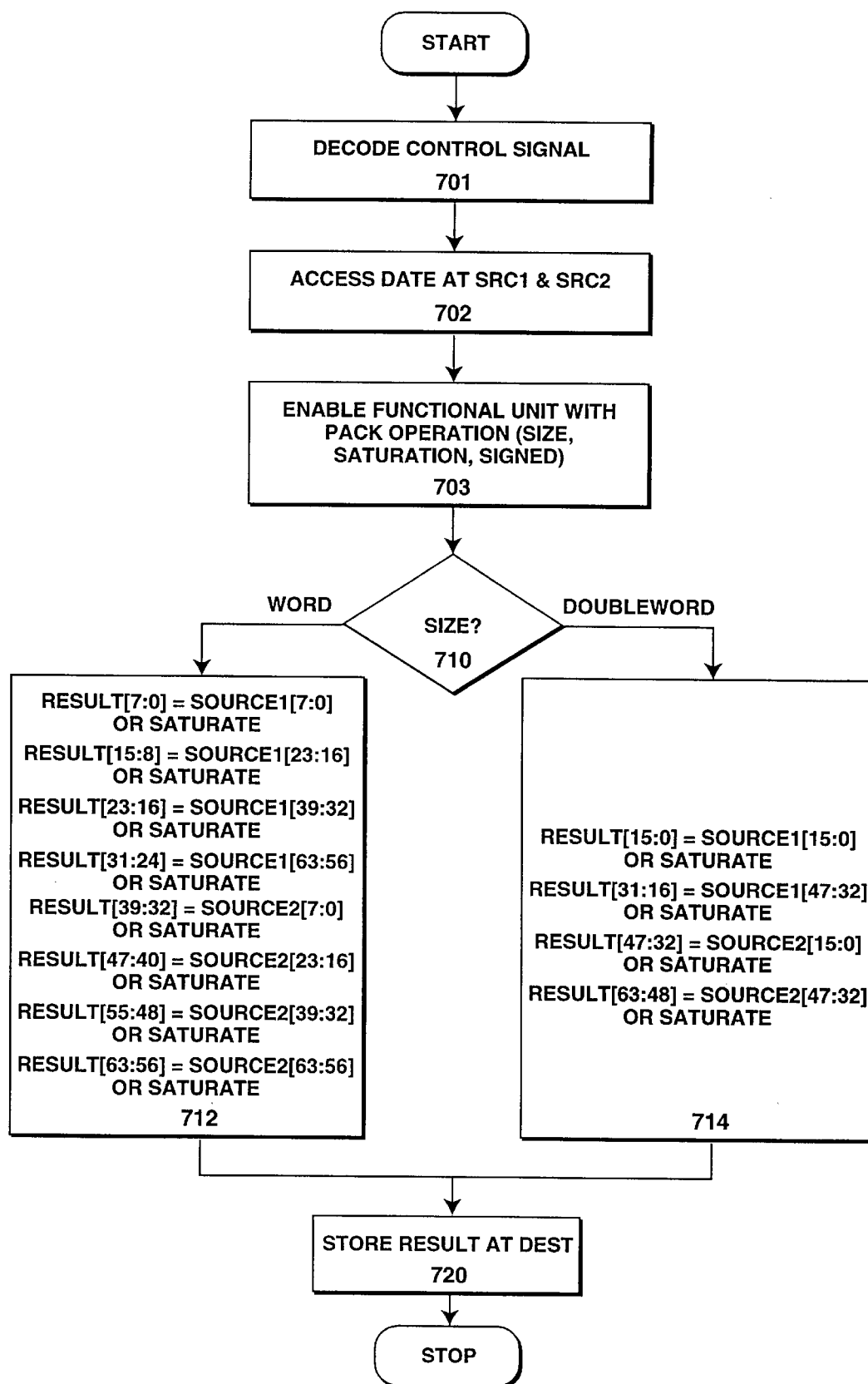


Figure 7

U.S. Patent

Oct. 6, 1998

Sheet 14 of 17

5,819,101

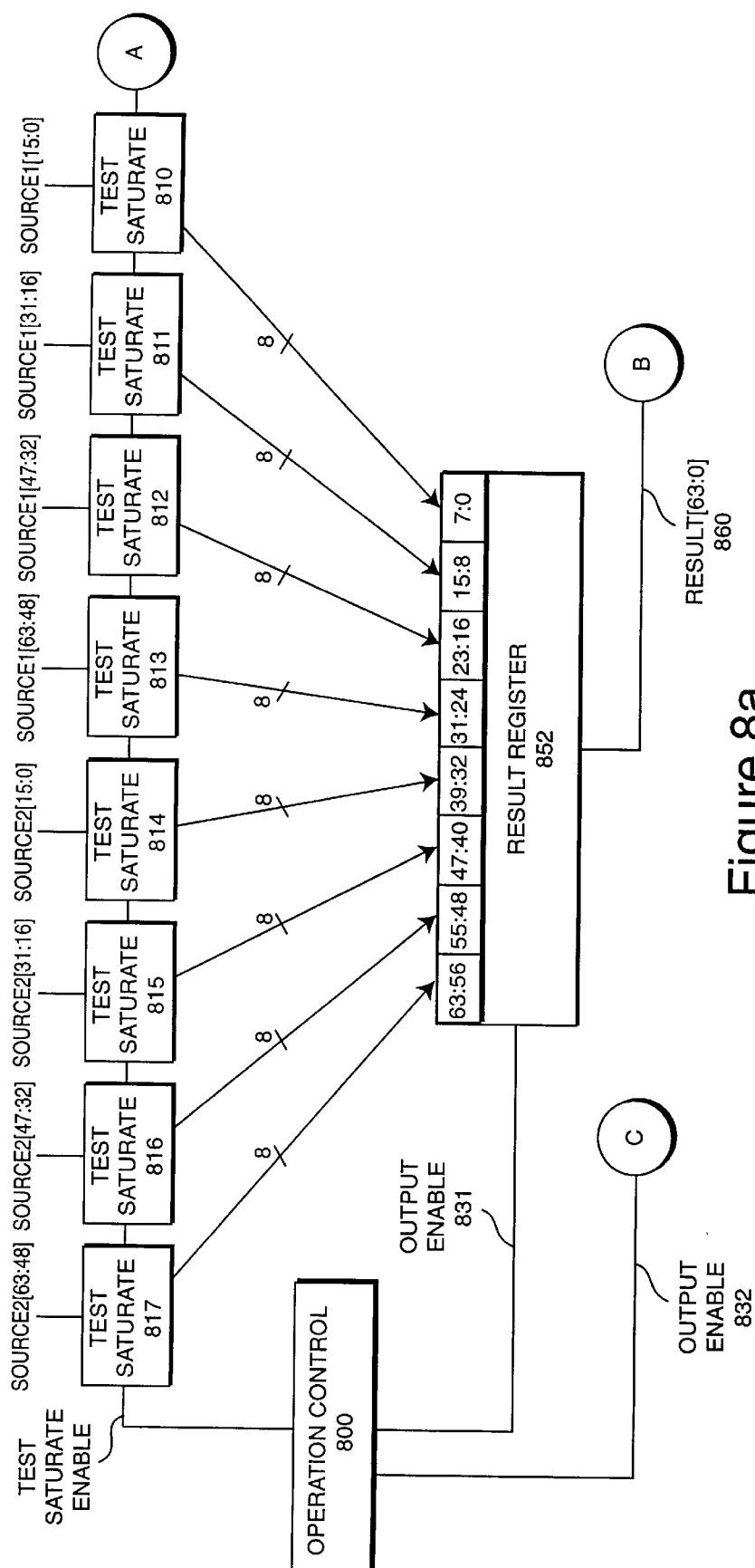


Figure 8a

U.S. Patent

Oct. 6, 1998

Sheet 15 of 17

5,819,101

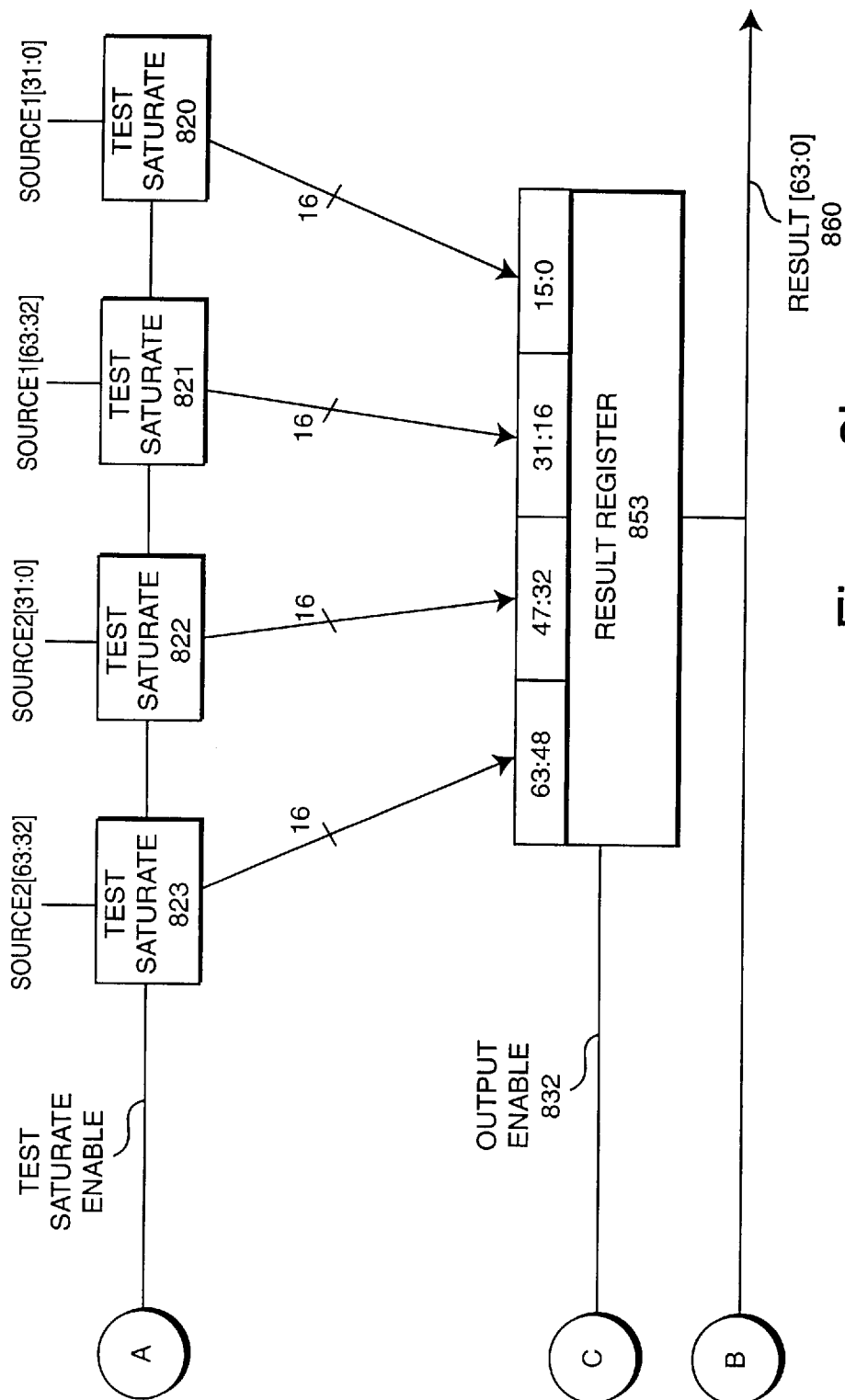


Figure 8b

U.S. Patent

Oct. 6, 1998

Sheet 16 of 17

5,819,101

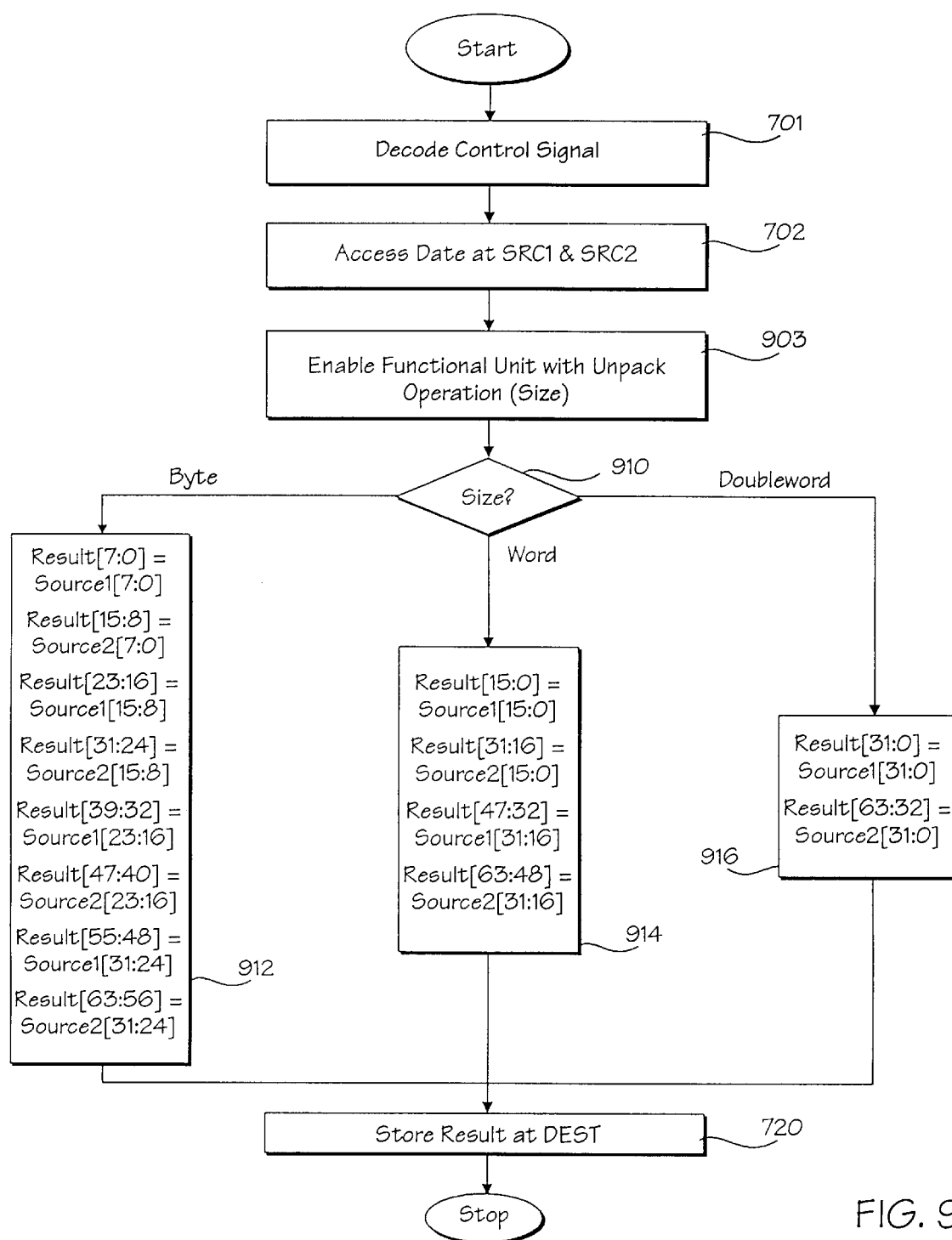


FIG. 9

U.S. Patent

Oct. 6, 1998

Sheet 17 of 17

5,819,101

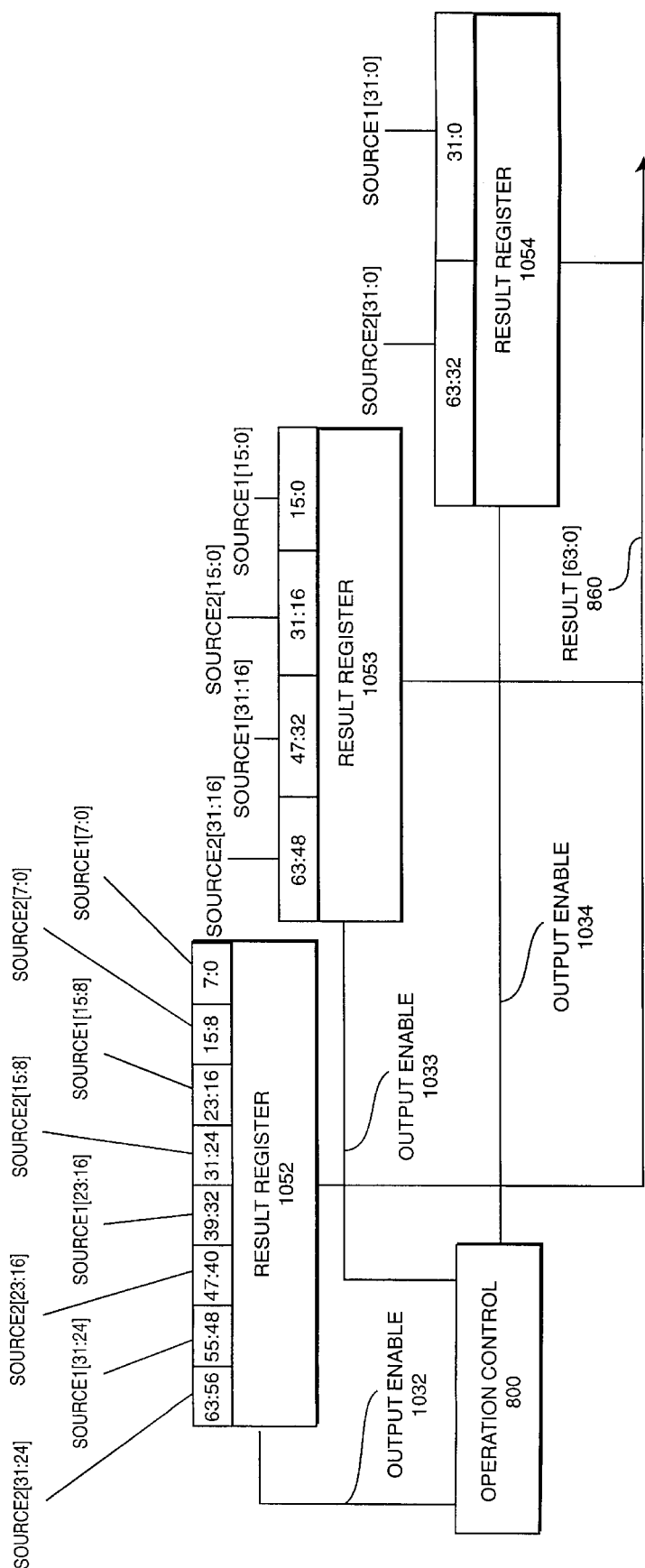


Figure 10

5,819,101

1

METHOD FOR PACKING A PLURALITY OF PACKED DATA ELEMENTS IN RESPONSE TO A PACK INSTRUCTION

This is a divisional of application Ser. No. 08/799,468, filed Feb. 13, 1997, which is a continuation of application Ser. No. 08/626,698 filed Apr. 2, 1996, now abandoned, which is a continuation of application Ser. No. 08/349,047 filed Dec. 2, 1994, now abandoned.

BACKGROUND OF THE INVENTION

1. Field of Invention

The present invention includes an apparatus and method of performing operations using a single control signal to manipulate multiple data elements. The present invention allows execution of move, pack and unpack operations on packed data types.

2. Description of Related Art

Today, most personal computer systems operate with one instruction to produce one result. Performance increases are achieved by increasing execution speed of instructions and the processor instruction complexity, and by performing multiple instructions in parallel; known as Complex Instruction Set Computer (CISC). Such processors as the Intel 80386™ microprocessor, available from Intel Corp. of Santa Clara, Calif., belong to the CISC category of processor.

Previous computer system architecture has been optimized to take advantage of the CISC concept. Such systems typically have data buses thirty-two bits wide. However, applications targeted at computer supported cooperation (CSC—the integration of teleconferencing with mixed media data manipulation), 2D/3D graphics, image processing, video compression/decompression, recognition algorithms and audio manipulation increase the need for improved performance. But, increasing the execution speed and complexity of instructions is only one solution.

One common aspect of these applications is that they often manipulate large amounts of data where only a few bits are important. That is, data whose relevant bits are represented in much fewer bits than the size of the data bus. For example, processors execute many operations on eight bit and sixteen bit data (e.g., pixel color components in a video image) but have much wider data busses and registers. Thus, a processor having a thirty-two bit data bus and registers, and executing one of these algorithms, can waste up to seventy-five percent of its data processing, carrying and storage capacity because only the first eight bits of data are important.

As such, what is desired is a processor that increases performance by more efficiently using the difference between the number of bits required to represent the data to be manipulated and the actual data carrying and storage capacity of the processor.

SUMMARY OF THE INVENTION

A processor having improved data manipulation operations is described.

A processor. The processor includes a first register for storing a first packed data, a decoder, and a functional unit. The decoder has a control signal input. The control signal input is for receiving a first control signal and a second control signal. The first control signal is for indicating a pack operation. The second control signal is for indicating an unpack operation. The functional unit is coupled to the decoder and the register. The functional unit is for perform-

2

ing the pack operation and the unpack operation using the first packed data. The processor also supports a move operation.

Although a great deal of detail has been included in the description and figures, the invention is defined by the scope of the claims. Only limitations found in those claims apply to the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not limitation, in the figures. Like references indicate similar elements.

FIG. 1 illustrates an embodiment of the computer system using the methods and apparatus of the present invention.

FIG. 2 illustrates an embodiment of the processor of the present invention.

FIG. 3 is a flow diagram illustrating the general steps used by the processor to manipulate data in the register file.

FIG. 4a illustrates memory data types.

FIG. 4b, FIG. 4c and FIG. 4d illustrate in-register integer data representations.

FIG. 5a illustrates packed data types.

FIG. 5b, FIG. 5c and FIG. 5d illustrate in-register packed data representations.

FIG. 6a illustrates a control signal format used in the computer system to indicate the use of packed data.

FIG. 6b illustrates a second control signal format that can be used in the computer to indicate the use of packed data or integer data.

FIG. 7 illustrates one embodiment of a method followed by a processor when performing a pack operation on packed data.

FIG. 8a illustrates a circuit capable of implementing a pack operation on packed byte data.

FIG. 8b illustrates a circuit capable of implementing a pack operation on packed word data.

FIG. 9 illustrates an embodiment of a method followed by a processor when performing an unpack operation on packed data.

FIG. 10 illustrates a circuit capable of implementing an unpack operation on packed data.

DESCRIPTION OF THE PREFERRED EMBODIMENT

OVERVIEW OF ONE EMBODIMENT OF THE PRESENT INVENTION

A processor having move, pack, and unpack operations that operate on multiple data elements is described. In the following description, numerous specific details are set forth such as circuits, etc., in order to provide a thorough understanding of the present invention. In other instances, well-known structures and techniques have not been shown in detail in order not to unnecessarily obscure the present invention.

DEFINITIONS

To provide a foundation for understanding the description of the embodiments of the present invention, the following definitions are provided.

Bit X through Bit Y:

defines a subfield of binary number. For example, bit six through bit zero of the byte 00111010₂ (shown in base

5,819,101

3

two) represent the subfield 111010₂. The '2' following a binary number indicates base 2. Therefore, 1000₂ equals 8₁₀, while F₁₆ equals 15₁₀.

Rx: is a register. A register is any device capable of storing and providing data. Further functionality of a register is described below. A register is not necessarily part of the processor's package.

DEST: is a data address.

SRC1: is a data address.

SRC2: is a data address.

Result: is the data to be stored in the register addressed by DEST.

Source1: is the data stored in the register addressed by SRC1.

Source2: is the data stored in the register addressed by SRC2.

COMPUTER SYSTEM

Referring to FIG. 1, a computer system upon which an embodiment of the present invention can be implemented is shown as computer system 100. Computer system 100 comprises a bus 101, or other communications hardware and software, for communicating information, and a processor 109 coupled with bus 101 for processing information. Computer system 100 further comprises a random access memory (RAM) or other dynamic storage device (referred to as main memory 104), coupled to bus 101 for storing information and instructions to be executed by processor 109. Main memory 104 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 109. Computer system 100 also comprises a read only memory (ROM) 106, and/or other static storage device, coupled to bus 101 for storing static information and instructions for processor 109. Data storage device 107 is coupled to bus 101 for storing information and instructions.

Furthermore, a data storage device 107, such as a magnetic disk or optical disk, and its corresponding disk drive, can be coupled to computer system 100. Computer system 100 can also be coupled via bus 101 to a display device 121 for displaying information to a computer user. Display device 121 can include a frame buffer, specialized graphics rendering devices, a cathode ray tube (CRT), and/or a flat panel display. An alphanumeric input device 122, including alphanumeric and other keys, is typically coupled to bus 101 for communicating information and command selections to processor 109. Another type of user input device is cursor control 123, such as a mouse, a trackball, a pen, a touch screen, or cursor direction keys for communicating direction information and command selections to processor 109, and for controlling cursor movement on display device 121. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), which allows the device to specify positions in a plane. However, this invention should not be limited to input devices with only two degrees of freedom.

Another device which may be coupled to bus 101 is a hard copy device 124 which may be used for printing instructions, data, or other information on a medium such as paper, film, or similar types of media. Additionally, computer system 100 can be coupled to a device for sound recording, and/or playback 125, such as an audio digitizer coupled to a microphone for recording information. Further, the device may include a speaker which is coupled to a digital to analog (D/A) converter for playing back the digitized sounds.

Also, computer system 100 can be a terminal in a computer network (e.g., a LAN). Computer system 100 would

4

then be a computer subsystem of a computer system including a number of networked devices. Computer system 100 optionally includes video digitizing device 126. Video digitizing device 126 can be used to capture video images that can be transmitted to others on the computer network.

Computer system 100 is useful for supporting computer supported cooperation (CSC—the integration of teleconferencing with mixed media data manipulation), 2D/3D graphics, image processing, video compression/decompression, recognition algorithms and audio manipulation.

PROCESSOR

FIG. 2 illustrates a detailed diagram of processor 109. Processor 109 can be implemented on one or more substrates using any of a number of process technologies, such as, BiCMOS, CMOS, and NMOS.

Processor 109 comprises a decoder 202 for decoding control signals and data used by processor 109. Data can then be stored in register file 204 via internal bus 205. As a matter of clarity, the registers of an embodiment should not be limited in meaning to a particular type of circuit. Rather, a register of an embodiment need only be capable of storing and providing data, and performing the functions described herein.

Depending on the type of data, the data may be stored in integer registers 201, registers 209, status registers 208, or instruction pointer register 211. Other registers can be included in the register file 204, for example, floating point registers. In one embodiment, integer registers 201 store thirty-two bit integer data. In one embodiment, registers 209 contains eight registers, R₀ 212a through R₇ 212h. Each register in registers 209 is sixty-four bits in length. R₁ 212a, R₂ 212b and R₃ 212c are examples of individual registers in registers 209. Thirty-two bits of a register in registers 209 can be moved into an integer register in integer registers 201. Similarly, an value in an integer register can be moved into thirty-two bits of a register in registers 209.

Status registers 208 indicate the status of processor 109. Instruction pointer register 211 stores the address of the next instruction to be executed. Integer registers 201, registers 209, status registers 208, and instruction pointer register 211 all connect to internal bus 205. Any additional registers would also connect to the internal bus 205.

In another embodiment, some of these registers can be used for two different types of data. For example, registers 209 and integer registers 201 can be combined where each register can store either integer data or packed data. In another embodiment, registers 209 can be used as floating point registers. In this embodiment, packed data can be stored in registers 209 or floating point data. In one embodiment, the combined registers are sixty-four bits in length and integers are represented as sixty-four bits. In this embodiment, in storing packed data and integer data, the registers do not need to differentiate between the two data types.

Functional unit 203 performs the operations carried out by processor 109. Such operations may include shifts, addition, subtraction and multiplication, etc. Functional unit 203 connects to internal bus 205. Cache 206 is an optional element of processor 109 and can be used to cache data and/or control signals from, for example, main memory 104. Cache 206 is connected to decoder 202, and is connected to receive control signal 207.

FIG. 3 illustrates the general operation of processor 109. That is, FIG. 3 illustrates the steps followed by processor

5,819,101

5

109 while performing an operation on packed data, performing an operation on unpacked data, or performing some other operation. For example, such operations include a load operation to load a register in register file 204 with data from cache 206, main memory 104, read only memory (ROM) 106, or data storage device 107. In one embodiment of the present invention, processor 109 supports most of the instructions supported by the Intel 80486™, available from Intel Corporation of Santa Clara, Calif. In another embodiment of the present invention, processor 109 supports all the operations supported by the Intel 80486™, available from Intel Corporation of Santa Clara, Calif. In another embodiment of the present invention, processor 109 supports all the operations supported by the Pentium™ processor, the Intel 80486™ processor, the 80386™ processor, the Intel 80286™ processor, and the Intel 8086™ processor, all available from Intel Corporation of Santa Clara, Calif. In another embodiment of the present invention, processor 109 supports all the operations supported in the IA™—Intel Architecture, as defined by Intel Corporation of Santa Clara, Calif. (see *Microprocessors*, Intel Data Books volume 1 and volume 2, 1992 and 1993, available from Intel of Santa Clara, Calif.). Generally, processor 109 can support the present instruction set for the Pentium™ processor, but can also be modified to incorporate future instructions, as well as those described herein. What is important is that general processor 109 can support previously used operations in addition to the operations described herein.

At step 301, the decoder 202 receives a control signal 207 from either the cache 206 or bus 101. Decoder 202 decodes the control signal to determine the operations to be performed.

Decoder 202 accesses the register file 204, or a location in memory, at step 302. Registers in the register file 204, or memory locations in the memory, are accessed depending on the register address specified in the control signal 207. For example, for an operation on packed data, control signal 207 can include SRC1, SRC2 and DEST register addresses. SRC1 is the address of the first source register. SRC2 is the address of the second source register. In some cases, the SRC2 address is optional as not all operations require two source addresses. If the SRC2 address is not required for an operation, then only the SRC1 address is used. DEST is the address of the destination register where the result data is stored. In one embodiment, SRC1 or SRC2 is also used as DEST. SRC1, SRC2 and DEST are described more fully in relation to FIG. 6a and FIG. 6b. The data stored in the corresponding registers is referred to as Source1, Source2, and Result respectively. Each of these data is sixty-four bits in length.

In another embodiment of the present invention, any one, or all, of SRC1, SRC2 and DEST, can define a memory location in the addressable memory space of processor 109. For example, SRC1 may identify a memory location in main memory 104 while SRC2 identifies a first register in integer registers 201, and DEST identifies a second register in registers 209. For simplicity of the description herein, references are made to the accesses to the register file 204, however, these accesses could be made to memory instead.

In another embodiment of the present invention, the operation code only includes two addresses, SRC1 and SRC2. In this embodiment, the result of the operation is stored in the SRC1 or SRC2 register. That is SRC1 (or SRC2) is used as the DEST. This type of addressing is compatible with previous CISC instructions having only two addresses. This reduces the complexity in the decoder 202. Note, in this embodiment, if the data contained in the SRC1

6

register is not to be destroyed, then that data must first be copied into another register before the execution of the operation. The copying would require an additional instruction. To simplify the description herein, the three address addressing scheme will be described (i.e. SRC1, SRC2, and DEST). However, it should be remembered that the control signal in one embodiment, may only include SRC1 and SRC2, and that SRC1 (or SRC2) identifies the destination register.

Where the control signal requires an operation, at step 303, functional unit 203 will be enabled to perform this operation on accessed data from register file 204. Once the operation has been performed in functional unit 203, at step 304, the result is stored back into register file 204 according to requirements of control signal 207.

DATA AND STORAGE FORMATS

FIG. 4a illustrates some of the data formats as may be used in the computer system of FIG. 1. These data formats are fixed point. Processor 109 can manipulate these data formats. Multimedia algorithms often use these data formats. A byte 401 contains eight bits of information. A word 402 contains sixteen bits of information, or two bytes. A doubleword 403 contains thirty-two bits of information, or four bytes. Thus, processor 109 executes control signals that may operate on any one of these memory data formats.

In the following description, references to bit, byte, word, and doubleword subfields are made. For example, bit six through bit zero of the byte 00111010₂ (shown in base 2) represent the subfield 111010₂.

FIG. 4b through FIG. 4d illustrate in-register representations used in one embodiment of the present invention. For example, unsigned byte in-register representation 410 can represent data stored in a register in integer registers 201. In one embodiment, a register, in integer registers 201, is sixty-four bits in length. In another embodiment, a register, in integer registers 201, is thirty-two bits in length. For the simplicity of the description, the following describes sixty-four bit integer registers, however, thirty-two bit integer registers can be used.

Unsigned byte in-register representation 410 illustrates processor 109 storing a byte 401 in integer registers 201, the first eight bits, bit seven through bit zero, in that register are dedicated to the data byte 401. These bits are shown as {b}. To properly represent this byte, the remaining 56 bits must be zero. For an signed byte in-register representation 411, integer registers 201 store the data in the first seven bits, bit six through bit zero, to be data. The seventh bit represents the sign bit, shown as an {s}. The remaining bit sixty-three through bit eight are the continuation of the sign for the byte.

Unsigned word in-register representation 412 is stored in one register of integer registers 201. Bit fifteen through bit zero contain an unsigned word 402. These bits are shown as {w}. To properly represent this word, the remaining bit sixty-three through bit sixteen must be zero. A signed word 402 is stored in bit fourteen through bit zero as shown in the signed word in-register representation 413. The remaining bit sixty-three through bit fifteen is the sign field.

A doubleword 403 can be stored as an unsigned doubleword in-register representation 414 or a signed doubleword in-register representation 415. Bit thirty-one through bit zero of an unsigned doubleword in-register representation 414 are the data. These bits are shown as {d}. To properly represent this unsigned doubleword, the remaining bit sixty-three through bit thirty-two must be zero. Integer registers 201 stores a signed doubleword in-register representation

5,819,101

7

415 in its bit thirty through bit zero; the remaining bit sixty-three through bit thirty-one are the sign field.

As indicated by the above FIG. **4b** through FIG. **4d**, storage of some data types in a sixty-four bit wide register is an inefficient method of storage. For example, for storage of an unsigned byte in-register representation **410** bit sixty-three through bit eight must be zero, while only bit seven through bit zero may contain non-zero bits. Thus, a processor storing a byte in a sixty-four bit register uses only 12.5% of the register's capacity. Similarly, only the first few bits of operations performed by functional unit **203** will be important.

FIG. **5a** illustrates the data formats for packed data. Each packed data includes more than one independent data element. Three packed data formats are illustrated; packed byte **501**, packed word **502**, and packed doubleword **503**. Packed byte, in one embodiment of the present invention, is sixty-four bits long containing eight data elements. Each data element is one byte long. Generally, a data element is an individual piece of data that is stored in a single register (or memory location) with other data elements of the same length. In one embodiment of the present invention, the number of data elements stored in a register is sixty-four bits divided by the length in bits of a data element.

Packed word **502** is sixty-four bits long and contains four word **402** data elements. Each word **402** data element contains sixteen bits of information.

Packed doubleword **503** is sixty-four bits long and contains two doubleword **403** data elements. Each doubleword **403** data element contains thirty-two bits of information.

FIG. **5b** through FIG. **5d** illustrate the in-register packed data storage representation. Unsigned packed byte in-register representation **510** illustrates the storage of packed byte **501** in one of the registers R_0 **212a** through R_n **212af**. Information for each byte data element is stored in bit seven through bit zero for byte zero, bit fifteen through bit eight for byte one, bit twenty-three through bit sixteen for byte two, bit thirty-one through bit twenty-four for byte three, bit thirty-nine through bit thirty-two for byte four, bit forty-seven through bit forty for byte five, bit fifty-five through bit forty-eight for byte six and bit sixty-three through bit fifty-six for byte seven. Thus, all available bits are used in the register. This storage arrangement increases the storage efficiency of the processor. As well, with eight data elements accessed, one operation can now be performed on eight data elements simultaneously. Signed packed byte in-register representation **511** is similarly stored in a register in registers **209**. Note that only the eighth bit of every byte data element is the necessary sign bit; other bits may or may not be used to indicate sign.

Unsigned packed word in-register representation **512** illustrates how word three through word zero are stored in one register of registers **209**. Bit fifteen through bit zero contain the data element information for word zero, bit thirty-one through bit sixteen contain the information for data element word one, bit forty-seven through bit thirty-two contain the information for data element word two and bit sixty-three through bit forty-eight contain the information for data element word three. Signed packed word in-register representation **513** is similar to the unsigned packed word in-register representation **512**. Note that only the sixteenth bit of each word data element contains the necessary sign indicator.

Unsigned packed doubleword in-register representation **514** shows how registers **209** store two doubleword data elements. Doubleword zero is stored in bit thirty-one

8

through bit zero of the register. Doubleword one is stored in bit sixty-three through bit thirty-two of the register. Signed packed doubleword in-register representation **515** is similar to unsigned packed doubleword in-register representation **514**. Note that the necessary sign bit is the thirty-second bit of the doubleword data element.

As mentioned previously, registers **209** may be used for both packed data and integer data. In this embodiment of the present invention, the individual programming processor **109** may be required to track whether an addressed register, R_1 **212a** for example, is storing packed data or simple integer/fixed point data. In an alternative embodiment, processor **109** could track the type of data stored in individual registers of registers **209**. This alternative embodiment could then generate errors if, for example, a packed addition operation were attempted on simple/fixed point integer data.

CONTROL SIGNAL FORMATS

The following describes one embodiment of control signal formats used by processor **109** to manipulate packed data. In one embodiment of the present invention, control signals are represented as thirty-two bits. Decoder **202** may receive control signal **207** from bus **101**. In another embodiment, decoder **202** can also receive such control signals from cache **206**.

FIG. **6a** illustrates a general format for a control signal operating on packed data. Operation field OP **601**, bit thirty-one through bit twenty-six, provides information about the operation to be performed by processor **109**; for example, packed addition, packed subtraction, etc. SRC1 **602**, bit twenty-five through twenty, provides the source register address of a register in registers **209**. This source register contains the first packed data, Source1, to be used in the execution of the control signal. Similarly, SRC2 **603**, bit nineteen through bit fourteen, contains the address of a register in registers **209**. This second source register contains the packed data, Source2, to be used during execution of the operation. DEST **605**, bit five through bit zero, contains the address of a register in registers **209**. This destination register will store the result packed data, Result, of the packed data operation.

Control bits SZ **610**, bit twelve and bit thirteen, indicates the length of the data elements in the first and second packed data source registers. If SZ **610** equals 01_2 , then the packed data is formatted as packed byte **501**. If SZ **610** equals 10_2 , then the packed data is formatted as packed word **502**. SZ **610** equaling 00_2 or 11_2 is reserved, however, in another embodiment, one of these values could be used to indicate packed doubleword **503**.

Control bit T **611**, bit eleven, indicates whether the operation is to be carried out with saturate mode. If T **611** equals one, then a saturating operation is performed. If T **611** equals zero, then a nonsaturating operation is performed. Saturating operations will be described later.

Control bit S **612**, bit ten, indicates the use of a signed operation. If S **612** equals one, then a signed operation is performed. If S **612** equals zero, then an unsigned operation is performed.

FIG. **6b** illustrates a second general format for a control signal operating on packed data. This format corresponds with the general integer opcode format described in the "Pentium™ Processor Family User's Manual," available from Intel Corporation, Literature Sales, P.O. Box 7641, Mt. prospect, Ill., 60056-7641. Note that OP **601**, SZ **610**, T **611**, and S **612** are all combined into one large field. For some control signals, bits three through five are SRC1 **602**. In one

5,819,101

9

embodiment, where there is a SRC1 602 address, then bits three through five also correspond to DEST 605. In an alternate embodiment, where there is a SRC2 603 address, then bits zero through two also correspond to DEST 605. For other control signals, like a packed shift immediate operation, bits three through five represent an extension to the opcode field. In one embodiment, this extension allows a programmer to include an immediate value with the control signal, such as a shift count value. In one embodiment, the immediate value follows the control signal. This is described in more detail in the "Pentium™ Processor Family User's Manual," in appendix F, pages F-1 through F-3. Bits zero through two represent SRC2 603. This general format allows register to register, memory to register, register by memory, register by register, register by immediate, register to memory addressing. Also, in one embodiment, this general format can support integer register to register, and register to integer register addressing.

DESCRIPTION OF SATURATE/UNSATURATE

As mentioned previously, T 611 indicates whether operations optionally saturate. Where the result of an operation, with saturate enabled, overflows or underflows the range of the data, the result will be clamped. Clamping means setting the result to a maximum or minimum value should a result exceed the range's maximum or minimum value. In the case of underflow, saturation clamps the result to the lowest value in the range and in the case of overflow, to the highest value. The allowable range for each data format is shown in Table 1.

TABLE 1

| Data Format | Minimum Value | Maximum Value |
|---------------------|------------------|--------------------|
| Unsigned Byte | 0 | 255 |
| Signed Byte | -128 | 127 |
| Unsigned Word | 0 | 65535 |
| Signed Word | -32768 | 32767 |
| Unsigned Doubleword | 0 | 2 ⁶⁴ -1 |
| Signed Doubleword | -2 ⁶³ | 2 ⁶³ -1 |

As mentioned above, T 611 indicates whether saturating operations are being performed. Therefore, using the unsigned byte data format, if an operation's result=258 and saturation was enabled, then the result would be clamped to 255 before being stored into the operation's destination register. Similarly, if an operation's result=-32999 and processor 109 used signed word data format with saturation enabled, then the result would be clamped to -32768 before being stored into the operation's destination register.

DATA MANIPULATION OPERATIONS

In one embodiment of the present invention, the performance of multimedia applications is improved by not only supporting a standard CISC instruction set (unpacked data operations), but by supporting operations on packed data. Such packed data operations can include an addition, a subtraction, a multiplication, a compare, a shift, an AND, and an XOR. However, to take full advantage of these operations, it has been determined that data manipulation operations should be included. Such data manipulation operations can include a move, a pack, and an unpack. Move, pack and unpack facilitate the execution of the other operations by generating packed data in formats that allow for easier use by programmers.

For further background on the other packed operations, "A Microprocessor Having a Compare Operation," filed on

10

Dec. 2, 1994, Ser. No. 08/349,040, now abandoned, "A Microprocessor Having a Multiply Operation," filed on Dec. 1, 1994, Ser. No. 08/349,559, now abandoned, "A Novel Processor Having Shift Operations," filed on Dec. 1, 1994, Ser. No. 08/349,730, now abandoned, "A Method and Apparatus Using Packed Data in a Processor," filed on Dec. 30, 1993, Ser. No. 08/176,123, now abandoned, and "A Method and Apparatus Using Novel Operations in a Processor," filed on Dec. 30, 1993, Ser. No. 08/175,772, now abandoned, all assigned to the assignee of the present invention.

MOVE OPERATION

The move operation transfers data to or from registers 209. In one embodiment, SRC2 603 is the address containing the source data and DEST 605 is the address where the data is to be transferred. In this embodiment, SRC1 602 would not be used. In another embodiment, SRC1 602 is DEST 605.

For the purposes of the explanation of the move operation, a distinction is drawn between a register and a memory location. Registers are found in register file 204 while memory can be, for example, in cache 206, main memory 104, ROM 106, data storage device 107.

The move operation can move data from memory to registers 209, from registers 209 to memory, and from a register in registers 209 to a second register in registers 209. In one embodiment, packed data is stored in different registers than those used to store integer data. In this embodiment, the move operation can move data from integer registers 201 to registers 209. For example, in processor 109, if packed data is stored in registers 209 and integer data is stored in integer registers 201, then a move instruction can be used to move data from integer registers 201 to registers 209, and vice versa.

In one embodiment, when a memory address is indicated for the move, the eight bytes of data at the memory location (the memory location indicating the least significant byte) are loaded to a register in registers 209 or stored from that register. When a register in registers 209 is indicated, the contents of that register are moved to or loaded from a second register in registers 209. If the integer registers 201 are sixty-four bits in length, and an integer register is specified, then the eight bytes of data in that integer register are loaded to a register in registers 209 or stored from that register.

In one embodiment, integers are represented as thirty-two bits. When a move operation is performed from registers 209 to integer registers 201, then only the low thirty-two bits of the packed data are moved to the specified integer register. In one embodiment, the high order thirty-two bits are zeroed. Similarly, only the low thirty-two bits of a register in registers 209 are loaded when a move is executed from integer registers 201 to registers 209. In one embodiment, processor 109 supports a thirty-two bit move operation between a register in registers 209 and memory. In another embodiment, a move of only thirty-two bits is performed on the high order thirty-two bits of packed data.

PACK OPERATION

In one embodiment of the present invention, the SRC1 602 register contains data (Source1), the SRC2 603 register contains the data (Source2), and DEST 605 register will contain the result data (Result) of the operation. That is, parts of Source1 and parts of Source2 will be packed together to generate Result.

In one embodiment, a pack operation converts packed words (or doublewords) into packed bytes (or words) by

5,819,101

11

packing the low order bytes (or words) of the source packed words (or doublewords) into the bytes (or words) of the Result. In one embodiment, the pack operation converts quad packed words into packed doublewords. This operation can be optionally performed with signed data. Further, this operation can be optionally performed with saturate.

FIG. 7 illustrates one embodiment of a method of performing a pack operation on packed data. This embodiment can be implemented in the processor 109 of FIG. 2.

At step 701, decoder 202 decodes control signal 207 received by processor 109. Thus, decoder 202 decodes: the operation code for the appropriate pack operation; SRC1 602, SRC2 603 and DEST 605 addresses in registers 209; saturate/unsaturate, signed/unsigned, and length of the data elements in the packed data. As mentioned previously, SRC1 602 (or SRC2 603) can be used as DEST 605.

At step 702, via internal bus 205, decoder 202 accesses registers 209 in register file 204 given the SRC1 602 and SRC2 603 addresses. Registers 209 provides functional unit 203 with the packed data stored in the SRC1 602 register (Source1), and the packed data stored in SRC2 603 register (Source2). That is, registers 209 communicate the packed data to functional unit 203 via internal bus 205.

At step 703, decoder 202 enables functional unit 203 to perform the appropriate pack operation. Decoder 202 further communicates, via internal bus 205, saturate and the size of the data elements in Source1 and Source2. Saturate is optionally used to maximize the value of the data in the result data element. If the value of the data elements in Source1 or Source2 are greater than or less than the range of values that the data elements of Result can represent, then the corresponding result data element is set to its highest or lowest value. For example, if signed values in the word data elements of Source1 and Source2 are smaller than 0x80 (or 0x8000 for doublewords), then the result byte (or word) data elements are clamped to 0x80 (or 0x8000 for doublewords). If signed values in word data elements of Source1 and Source 2 are greater than 0x7F (or 0x7FFF for doublewords), then the result byte (or word) data elements are clamped to 0x7F (or 0x7FFF).

At step 710, the size of the data element determines which step is to be executed next. If the size of the data elements is sixteen bits (packed word 502 data), then functional unit 203 performs step 712. However, if the size of the data elements in the packed data is thirty-two bits (packed doubleword 503 data), then functional unit 203 performs step 714.

12

performed. Source1 bits seven through zero are Result bits seven through zero. Source1 bits twenty-three through sixteen are Result bits fifteen through eight. Source1 bits thirty-nine through thirty-two are Result bits twenty-three through sixteen. Source1 bits sixty-three through fifty-six are Result bits thirty-one through twenty-four. Source2 bits seven through zero are Result bits thirty-nine through thirty-two. Source2 bits twenty-three through sixteen are Result bits forty-seven through forty. Source2 bits thirty-nine through thirty-two are Result bits fifty-five through forty-eight. Source2 bits sixty-three through fifty-six are Result bits thirty-one through twenty-four. If saturate is set, then the high order bits of each word are tested to determine whether the Result data element should be clamped.

Assuming the size of the source data elements is thirty-two bits, then step 714 is executed. In step 714, the following is performed. Source1 bits fifteen through zero are Result bits fifteen through zero. Source1 bits forty-seven through thirty-two are Result bits thirty-one through sixteen. Source2 bits fifteen through zero are Result bits forty-seven through thirty-two. Source2 bits forty-seven through thirty-two are Result bits sixty-three through forty-eight. If saturate is set, then the high order bits of each doubleword are tested to determine whether the Result data element should be clamped.

In one embodiment, the packing of step 712 is performed simultaneously. However, in another embodiment, this packing is performed serially. In another embodiment, some of the packing is performed simultaneously and some is performed serially. This discussion also applies to the packing of step 714.

At step 720, the Result is stored in the DEST 605 register.

Table 2 illustrates the in-register representation of a pack unsigned word operation with no saturation. The first row of bits is the packed data representation of Source1. The second row of bits is the data representation of Source2. The third row of bits is the packed data representation of the Result. The number below each data element bit is the data element number. For example, Source1 data element three is 10000000₂.

TABLE 2

| Source1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| | 3 | | 2 | | 1 | | 0 |
| Source2 | | | | | | | |
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| | 3 | | 2 | | 1 | | 0 |
| Result | | | | | | | |
| 00000000 | 00000000 | 00000000 | 10001000 | 01010101 | 11111111 | 01110000 | 10001000 |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | | | | | | | 0 |

Assuming the size of the source data elements is sixteen bits, then step 712 is executed. In step 712, the following is

Table 3 illustrates the in-register representation of pack signed doubleword operation with saturation.

5,819,101

13

14

TABLE 3

| Source1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| 1 | | | | 0 | | | |
| Source2 | | | | | | | |
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| 1 | | | | 0 | | | |
| Result | | | | | | | |
| 11000000 | 00000000 | 10000000 | 00000000 | 01111111 | 11111111 | 10000000 | 00000000 |
| 3 | | 2 | | 1 | | 0 | |

PACK CIRCUITS

In one embodiment of the present invention, to achieve efficient execution of the pack operation parallelism is used. FIG. 8a and 8b illustrate one embodiment of a circuit that can perform a pack operation on packed data. The circuit can optionally perform the pack operation with saturation.

The circuit of FIG. 8a and 8b includes an operation control circuit 800, a result register 852, a result register 853, eight sixteen bit to eight bit test saturate circuits, and four thirty-two bit to sixteen bit test saturate circuits.

Operation control 800 receives information from the decoder 202 to enable a pack operation. Operation control 800 uses the saturate value to enable the saturation tests for each of the test saturate circuits. If the size of the source packed data is word packed data 503, then output enable 831 is set by operation control 800. This enables the output of output register 852. If the size of the source packed data is doubleword packed data 504, then output enable 832 is set by operation control 800. This enables the output of output register 853.

Each test saturate circuit can selectively test for saturation. If a test for saturation is disabled, then each test saturate circuit merely passes the low order bits through to a corresponding position in a result register. If a test for saturate is enabled, then each test saturate circuit tests the high order bits to determine if the result should be clamped.

Test saturate 810 through test saturate 817 have sixteen bit inputs and eight bit outputs. The eight bit outputs are the lower eight bits of the inputs, or optionally, are a clamped value (0x80, 0x7F, or 0xFF). Test saturate 810 receives Source1 bits fifteen through zero and outputs bits seven through zero for result register 852. Test saturate 811

15 852. Test saturate 813 receives Source1 bits sixty-three through forty-eight and outputs bits thirty-one through twenty-four for result register 852. Test saturate 814 receives Source2 bits fifteen through zero and outputs bits thirty-nine through thirty-two for result register 852. Test saturate 815
20 receives Source2 bits thirty-one through sixteen and outputs bits forty-seven through forty for result register 852. Test saturate 816 receives Source2 bits forty-seven through thirty-two and outputs bits fifty-five through forty-eight for result register 852. Test saturate 817 receives Source2 bits
25 sixty-three through forty-eight and outputs bits sixty-three through fifty-six for result register 852.

Test saturate 820 through test saturate 823 have thirty-two bit inputs and sixteen bit outputs. The sixteen bit outputs are the lower sixteen bits of the inputs, or optionally, are a clamped value (0x8000, 0x7FFF, or 0xFFFF). Test saturate
30 820 receives Source1 bits thirty-one through zero and outputs bits fifteen through zero for result register 853. Test saturate 821 receives Source1 bits sixty-three through thirty-two and outputs bits thirty-one through sixteen for result register 853. Test saturate 822 receives Source2 bits thirty-one through zero and outputs bits forty-seven through thirty-two for result register 853. Test saturate 823 receives
40 Source2 bits sixty-three through thirty-two and outputs bits sixty-three through forty-eight of result register 853.

For example, in Table 4, a pack word unsigned with no saturate is performed. Operation control 800 will enable result register 852 to output result[63:0] 860.

TABLE 4

| | | | | | | | | | | | | | | | |
|---------|--|-----|-----|----------|----------|----------|----------|-----|----------|-----|----------|----------|--|----------|--|
| Source1 | | | | | | | | | | | | | | | |
| ... | | | ... | | 00001110 | | 01110000 | | 00001110 | | 00001000 | | | | |
| 3 | | | 2 | | 1 | | 0 | | | | | | | | |
| Source2 | | | | | | | | | | | | | | | |
| ... | | | ... | | 00001110 | | 10000001 | | 00001110 | | 10000001 | | | | |
| 3 | | | 2 | | 1 | | 0 | | | | | | | | |
| Result | | | | | | | | | | | | | | | |
| ... | | ... | | 10000001 | | 10000001 | | ... | | ... | | 01110000 | | 00001000 | |
| 7 | | 6 | | 5 | | 4 | | 3 | | 2 | | 1 | | 0 | |

60

receives Source1 bits thirty-one through sixteen and outputs bits fifteen through eight for result register 852. Test saturate 812 receives Source1 bits forty-seven through thirty-two and outputs bits twenty-three through sixteen for result register

However, if a pack doubleword unsigned with no saturate is performed, operation control 800 will enable result register 853 to output result[63:0] 860. Table 5 illustrates this result.

15

16

TABLE 5

| | | | | |
|-----|----------|----------|----------|----------|
| | | | | Source1 |
| ... | ... | 00001110 | 01000001 | 00001110 |
| | 1 | | | 0 |
| | | | | Source2 |
| ... | ... | 00001110 | 00000001 | 00001110 |
| | 1 | | | 0 |
| | | | | Result |
| ... | 00001110 | 10000001 | ... | 00001110 |
| 3 | | 2 | 1 | 0 |

UNPACK OPERATION

In one embodiment, an unpack operation interleaves the low order packed bytes, words or doublewords of two source packed data to generate result packed bytes, words, or doublewords.

FIG. 9 illustrates one embodiment of a method of performing an unpack operation on packed data. This embodiment can be implemented in the processor 109 of FIG. 2.

Step **701** and step **702** are executed first. At step **903**, decoder **202** enables functional unit **203** to perform the unpack operation. Decoder **202** communicates, via internal bus **205**, the size of the data elements in Source1 and Source2.

At step **910**, the size of the data element determines which step is to be executed next. If the size of the data elements is eight bits (packed byte **501** data), then functional unit **203** performs step **712**. However, if the size of the data elements in the packed data is sixteen bits (packed word **502** data), then functional unit **203** performs step **714**. However, if the size of the data elements in the packed data is thirty-two bits (packed doubled word **503** data), then functional unit **203** performs step **716**.

Assuming the size of the source data elements is eight bits, then step 712 is executed. In step 712, the following is performed. Source1 bits seven through zero are Result bits seven through zero. Source2 bits seven through zero are Result bits fifteen through eight. Source1 bits fifteen through eight are Result bits twenty-three through sixteen. Source2 bits fifteen through eight are Result bits thirty-one through

15 twenty-three through sixteen are Result bits forty-seven through forty. Source1 bits thirty-one through twenty-four are Result bits fifty-five through forty-eight Source2 bits thirty-one through twenty-four are Result bits sixty-three through fifty-six.

Assuming the size of the source data elements is sixteen bits, then step 714 is executed. In step 714, the following is performed. Source1 bits fifteen through zero are Result bits fifteen through zero. Source2 bits fifteen through zero are Result bits thirty-one through sixteen. Source1 bits thirty-one through sixteen are Result bits forty-seven through thirty-two. Source2 bits thirty-one through sixteen are Result bits sixty-three through forty-eight.

Assuming the size of the source data elements is thirty-
 30 two bits, then step **716** is executed. In step **716**, the following
 is performed. Source1 bits thirty-one through zero are Result
 bits thirty-one through zero. Source2 bits thirty-one through
 zero are Result bits sixty-three through thirty-two.

In one embodiment, the unpacking of step 712 is performed simultaneously. However, in another embodiment, this unpacking is performed serially. In another embodiment, some of the unpacking is performed simultaneously and some is performed serially. This discussion also applies to the unpacking of step 714 and step 716.

At step **720**, the Result is stored in the DEST **605** register.

Table 6 illustrates the in-register representation of an unpack byte operation.

TABLE 6

Source1

| | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Source2

| | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Result

| | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 11110011 | 10000000 | 00000000 | 01110000 | 10001110 | 10001111 | 10001000 | 10001000 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

twenty-four. Source1 bits twenty-three through sixteen are Result bits thirty-nine through thirty-two. Source2 bits

Table 7 illustrates the in-register representation of an unpack word operation.

TABLE 7

| | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| | | | | | | | Source1 |
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| | | | 3 | 2 | 1 | 0 | |

5,819,101

17

18

TABLE 7-continued

| Source2 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| 3 | | 2 | | 1 | | | 0 |
| Result | | | | | | | |
| 11110011 | 00000000 | 10000000 | 01110000 | 10001110 | 10001000 | 10001111 | 10001000 |
| 3 | | 2 | | 1 | | | 0 |

10

Table 8 illustrates the in-register representation of an unpack doubleword operation.

are bits thirty-one through twenty-four for result register **1052**. Source1 bits twenty-three through sixteen are bits

TABLE 8

| Source1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| | | | 1 | | | | 0 |
| Source2 | | | | | | | |
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| | | | 1 | | | | 0 |
| Result | | | | | | | |
| 11110011 | 00000000 | 10001110 | 10001000 | 10000000 | 01110000 | 10001111 | 10001000 |
| | | | 1 | | | | 0 |

UNPACK CIRCUITS

In one embodiment of the present invention, to achieve efficient execution of the unpack operation parallelism is used. FIG. 10 illustrates one embodiment of a circuit that can perform an unpack operation on packed data.

The circuit of FIG. 10 includes the operation control circuit **800**, a result register **1052**, a result register **1053**, and a result register **1054**.

Operation control **800** receives information from the decoder **202** to enable an unpack operation. If the size of the source packed data is byte packed data **502**, then output enable **1032** is set by operation control **800**. This enables the output of result register **1052**. If the size of the source packed data is word packed data **503**, then output enable **1033** is set by operation control **800**. This enables the output of output register **1053**. If the size of the source packed data is doubleword packed data **504**, then output enable **1034** is set by operation control **800**. This enables the output of output register **1054**.

Result register **1052** has the following inputs. Source1 bits seven through zero are bits seven through zero for result

thirty-nine through thirty-two for result register **1052**. Source2 bits twenty-three through sixteen are bits forty-seven through forty for result register **1052**. Source1 bits thirty-one through twenty-four are bits fifty-five through forty-eight for result register **1052**. Source2 bits thirty-one through twenty-four are bits sixty-three through fifty-six for result register **1052**.

Result register **1053** has the following inputs. Source1 bits fifteen through zero are bits fifteen through zero for result register **1053**. Source2 bits fifteen through zero are bits thirty-one through sixteen for result register **1053**. Source1 bits thirty-one through sixteen are bits forty-seven through thirty-two for result register **1053**. Source2 bits thirty-one through sixteen are bits sixty-three through forty-eight of result register **853**.

Result register **1054** has the following inputs. Source1 bits thirty-one through zero are bits thirty-one through zero for result register **1054**. Source2 bits thirty-one through zero are bits sixty-three through thirty-two of result register **1054**.

For example, in Table 9, an unpack word operation is performed. Operation control **800** will enable result register **1053** to output result[63:0] **860**.

TABLE 9

| Source1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| ... | ... | 00001110 | 01110000 | 00001110 | 00001000 | | |
| 3 | 2 | | 1 | | | | 0 |
| Source2 | | | | | | | |
| ... | ... | 00001110 | 00000001 | 00001110 | 10000001 | | |
| 3 | 2 | | 1 | | | | 0 |
| Result | | | | | | | |
| 00001110 | 00000001 | 00001110 | 01110000 | 00001110 | 10000001 | 00001110 | 00001000 |
| 3 | 2 | | 1 | | | | 0 |

register **1052**. Source2 bits seven through zero are bits fifteen through eight for result register **1052**. Source1 bits fifteen through eight are bits twenty-three through sixteen for result register **1052**. Source 2 bits fifteen through eight

However, if an unpack doubleword is performed, operation control **800** will enable result register **1054** to output result[63:0] **860**. Table 10 illustrates this result.

5,819,101

19

20

TABLE 10

| Source1 | | | | |
|----------|----------|----------|----------|----------|
| ... | 00001110 | 01000001 | 00001110 | 00001000 |
| 1 | | | | 0 |
| Source2 | | | | |
| ... | 00001110 | 00000001 | 00001110 | 10000001 |
| 1 | | | | 0 |
| Result | | | | |
| 00001110 | 00000001 | 00001110 | 10000001 | 00001110 |
| 1 | | | | 0 |

Therefore, the move, pack and unpack operations can manipulate multiple data elements. In prior art processors, to perform these types of manipulations, multiple separate operations would be needed to perform a single packed move, pack or unpack operation. The data lines for the packed data operations, in one embodiment, all carry relevant data. This leads to a higher performance computer system.

What is claimed is:

1. A method for manipulating packed data in a computer system comprising the computer implemented steps of:

- a) decoding a Single Instruction Multiple Data (SIMD) pack instruction, the instruction identifying a first and second packed data respectively including a first plurality of data elements and a second plurality of data elements, each data element consisting of a separate multiple bit data field, wherein each data element in the first plurality of data elements corresponds to a data element in the second plurality of data elements in a respective position; and

- b) simultaneously copying, in response to the pack instruction, a part of each data element in the first plurality of data elements and a part of each corresponding data element in the second plurality of data elements into a third packed data as a plurality of separate result data elements.

2. The method of claim 1, wherein the part is half of the bits in each data element in the first and second plurality of data elements.

3. The method of claim 2, wherein the part is either the low or the high order bits of each data element in the first and second plurality of data elements.

4. The method of claim 3, wherein the first plurality of data elements and the second plurality of data elements each include either two, four, or eight data elements.

5. The method of claim 4, wherein the parts copied from the first plurality of data elements are stored adjacent to each other in the plurality of result data elements.

6. The method of claim 5, wherein the parts copied from the first and second plurality of data elements are stored in the same order as the first and second plurality of data elements appear in the first and second packed data.

7. The method of claim 6, wherein all data elements in the first and second plurality of data elements are signed, and wherein all data elements in the third plurality of data elements are either signed or unsigned.

8. The method of claim 7, wherein all data elements of the plurality of result data elements are either saturated or unsaturated.

9. A computer implemented method for manipulating data elements in a first and second packed data in response to a Single Instruction Multiple Data (SIMD) pack instruction, the first and second packed data respectively including a first plurality of data elements and a second plurality of data elements, each data element consisting of a separate multiple bit data field, wherein each data element in the first plurality of data elements corresponds to a different element in the second plurality of data elements in a respective position, the method comprising the computer implemented steps of:

- a) decoding the SIMD pack instruction;
- b) reading the first packed data and reading the second packed data;
- c) simultaneously copying, in response to the pack instruction, a part of each data element in the first and second plurality of data elements into a third packed data sequence as a third plurality of separate data elements.

10. The method of claim 9, wherein the part is half of the bits in each data element in the first and second plurality of data elements.

11. The method of claim 10, wherein the part is either the low or the high order bits of each data element in the first and second plurality of data elements.

12. The method of claim 11, wherein the first plurality of data elements and the second plurality of data elements each include either two, four, or eight data elements.

13. The method of claim 12, wherein the parts copied from the first plurality of data elements are stored adjacent to each other in the plurality of result data elements.

14. The method of claim 13, wherein the parts copied from the first and second plurality of data elements are stored in the same order as the first and second plurality of data elements appear in the first and second packed data.

15. The method of claim 14, wherein all data elements in the first and second plurality of data elements are signed, and wherein all data elements in the third plurality of data elements are either signed or unsigned.

16. The method of claim 15, wherein all data elements of the plurality of result data elements are either saturated or unsaturated.

* * * * *

EXHIBIT 5



US005881275A

United States Patent

[19]

[11] **Patent Number:** **5,881,275****Peleg et al.**[45] **Date of Patent:** **Mar. 9, 1999**

[54] **METHOD FOR UNPACKING A PLURALITY OF PACKED DATA INTO A RESULT PACKED DATA**

[75] Inventors: **Alexander Peleg**, Carmelia; **Yaakov Yaari**, Haifa, both of Israel; **Millind Mittal**, South San Francisco; **Larry M. Mennemeier**, Boulder Creek, both of Calif.; **Benny Eitan**, Haifa, Israel

[73] Assignee: **Intel Corporation**, Santa Clara, Calif.

[21] Appl. No.: **799,468**

[22] Filed: **Feb. 13, 1997**

Related U.S. Application Data

[63] Continuation of Ser. No. 626,698, Apr. 2, 1996, abandoned, which is a continuation of Ser. No. 349,047, Dec. 2, 1994, abandoned.

[51] **Int. Cl.**⁶ **G06F 9/30**

[52] **U.S. Cl.** **395/564**; 395/376; 395/562; 395/800.32; 364/715.011

[58] **Field of Search** 395/376, 562, 395/564, 800.32; 364/715.011

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,268,995 12/1993 Diefendorff et al. 345/422

5,327,543 7/1994 Miura et al. 395/565
 5,390,135 2/1995 Lee et al. 364/749
 5,408,670 4/1995 Davies 395/800.16
 5,423,010 6/1995 Mizkami 341/60
 5,499,376 3/1996 Kay et al. 395/800.3

OTHER PUBLICATIONS

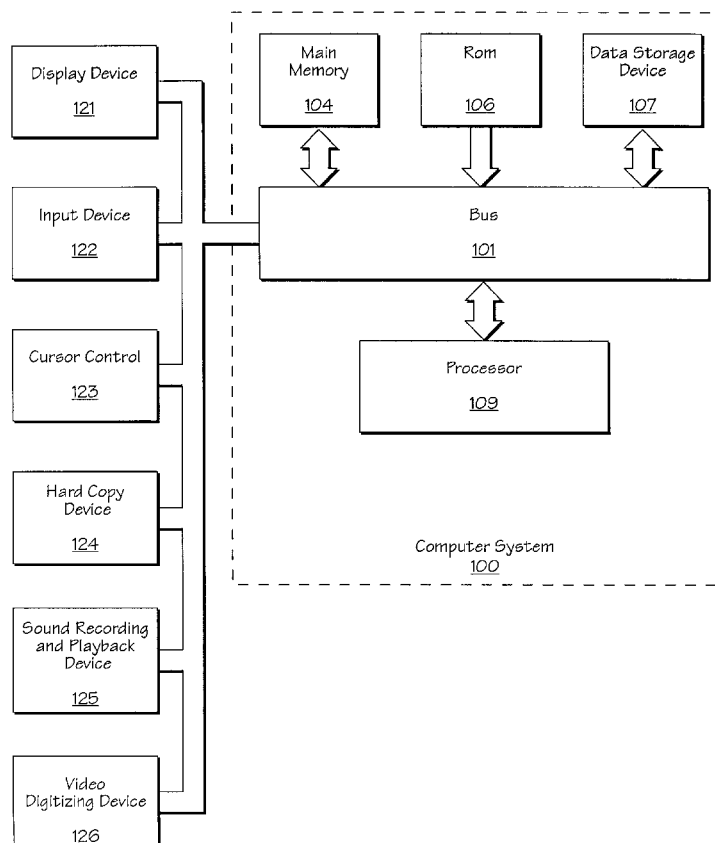
UltraSparc, Sparc Technology Business, Sep. 1994 pp. 1-8.
 MC88110 User's Manual, Motorola, 1991, p. 5:11-12.

Primary Examiner—Kenneth S. Kim

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A processor. The processor includes a first register for storing a first packed data, a decoder, and a functional unit. The decoder has a control signal input. The control signal input is for receiving a first control signal and a second control signal. The first control signal is for indicating a pack operation. The second control signal is for indicating an unpack operation. The functional unit is coupled to the decoder and the register. The functional unit is for performing the pack operation and the unpack operation using the first packed data. The processor also supports a move operation.

10 Claims, 17 Drawing Sheets

U.S. Patent

Mar. 9, 1999

Sheet 1 of 17

5,881,275

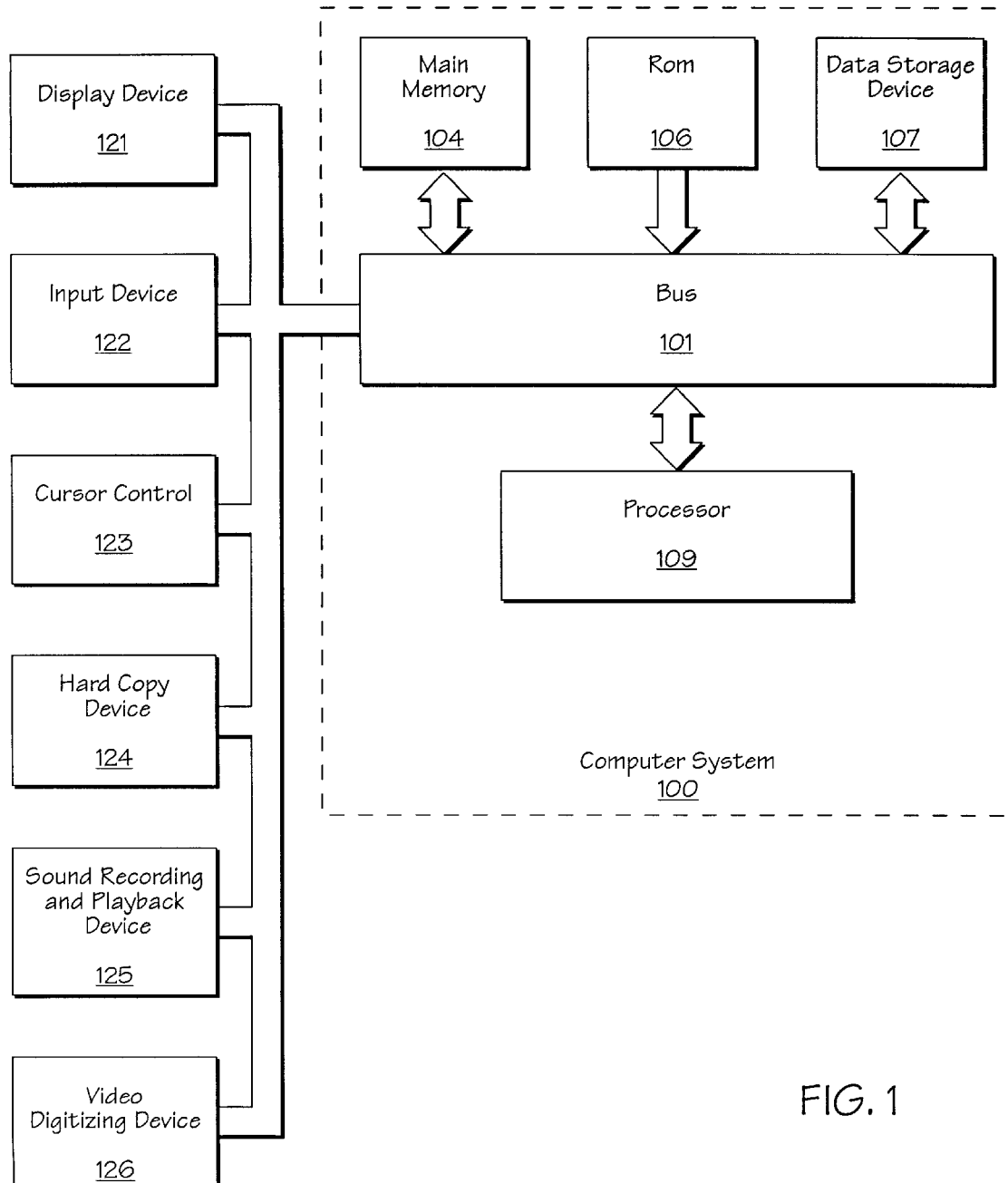


FIG. 1

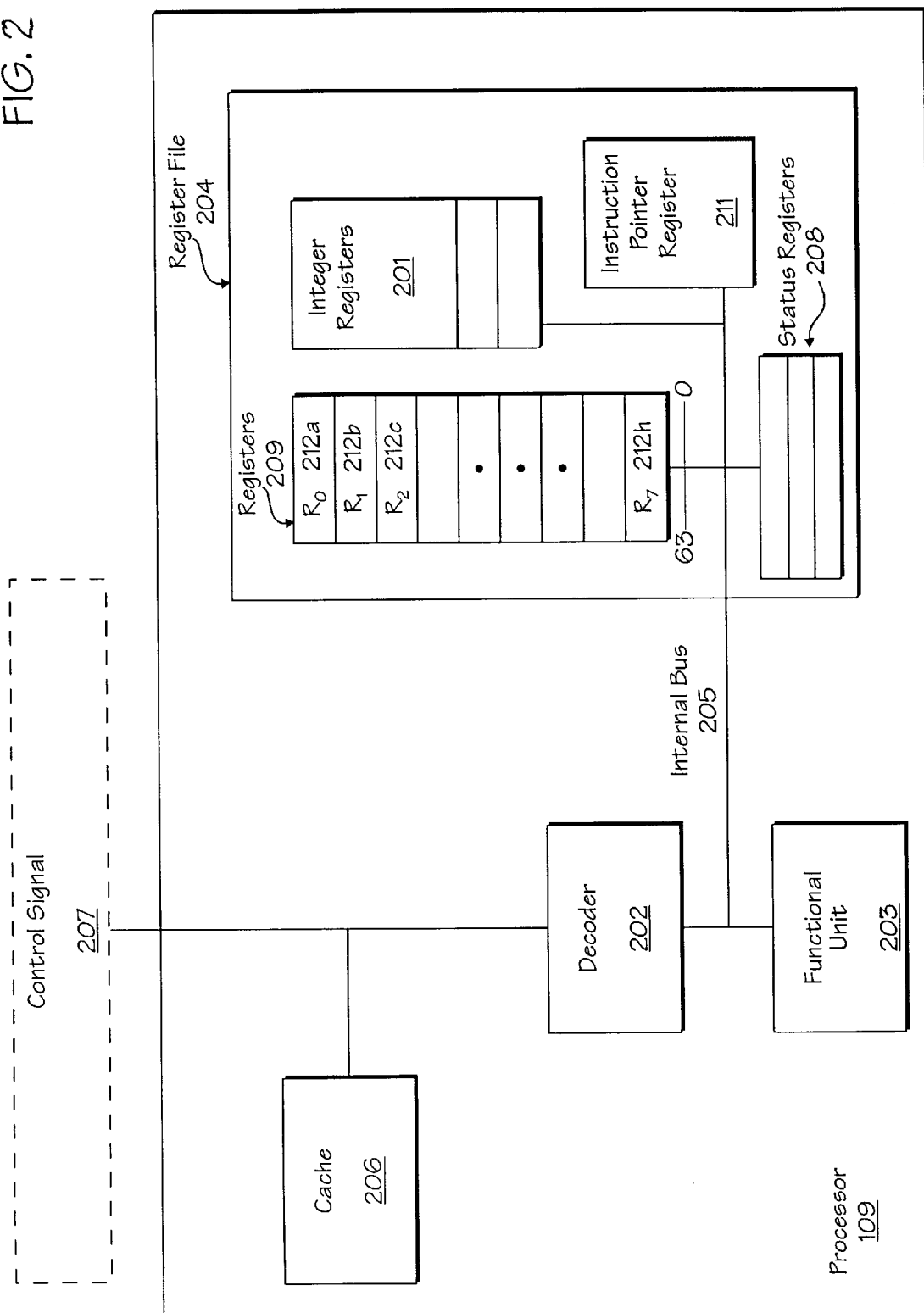
U.S. Patent

Mar. 9, 1999

Sheet 2 of 17

5,881,275

FIG. 2



U.S. Patent

Mar. 9, 1999

Sheet 3 of 17

5,881,275

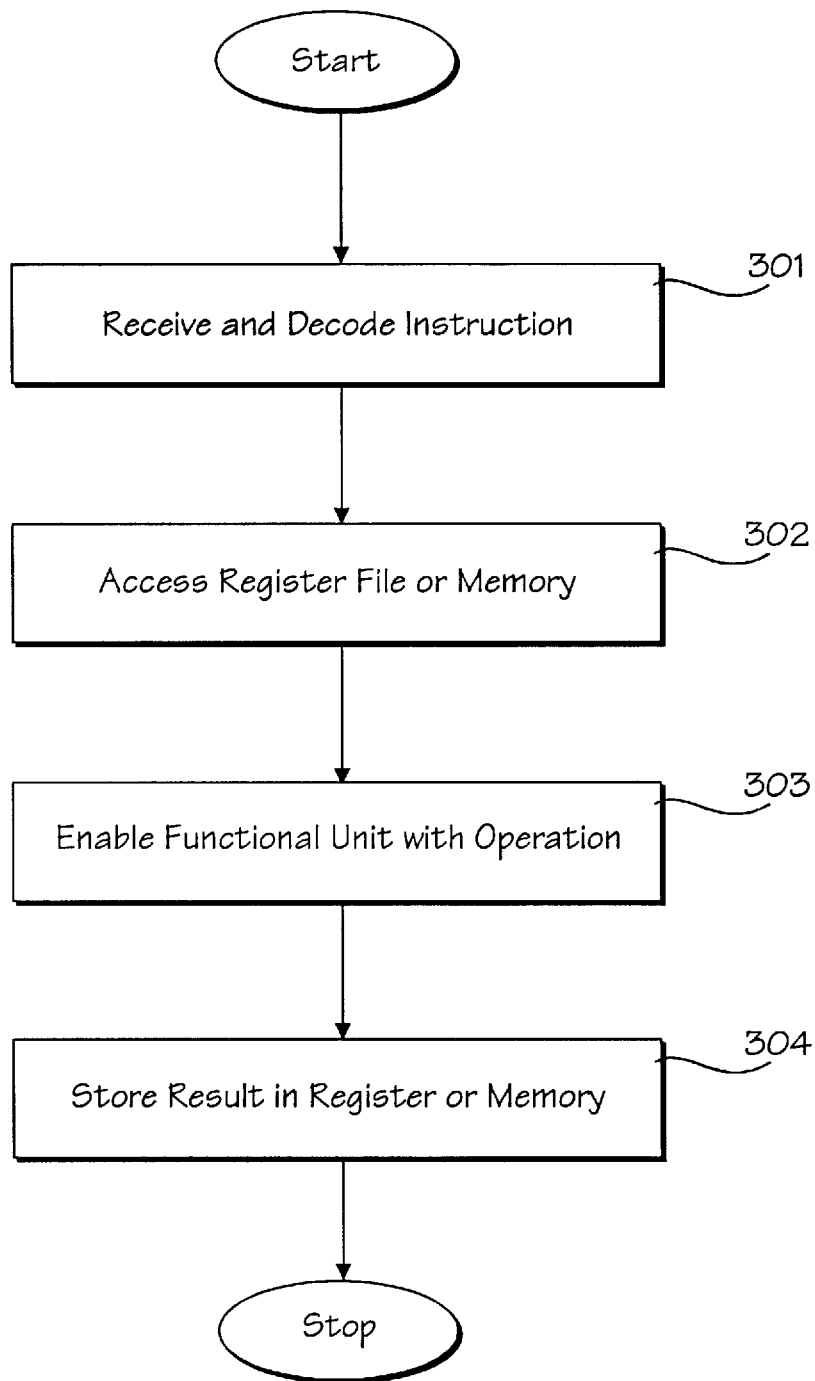


FIG. 3

U.S. Patent

Mar. 9, 1999

Sheet 4 of 17

5,881,275

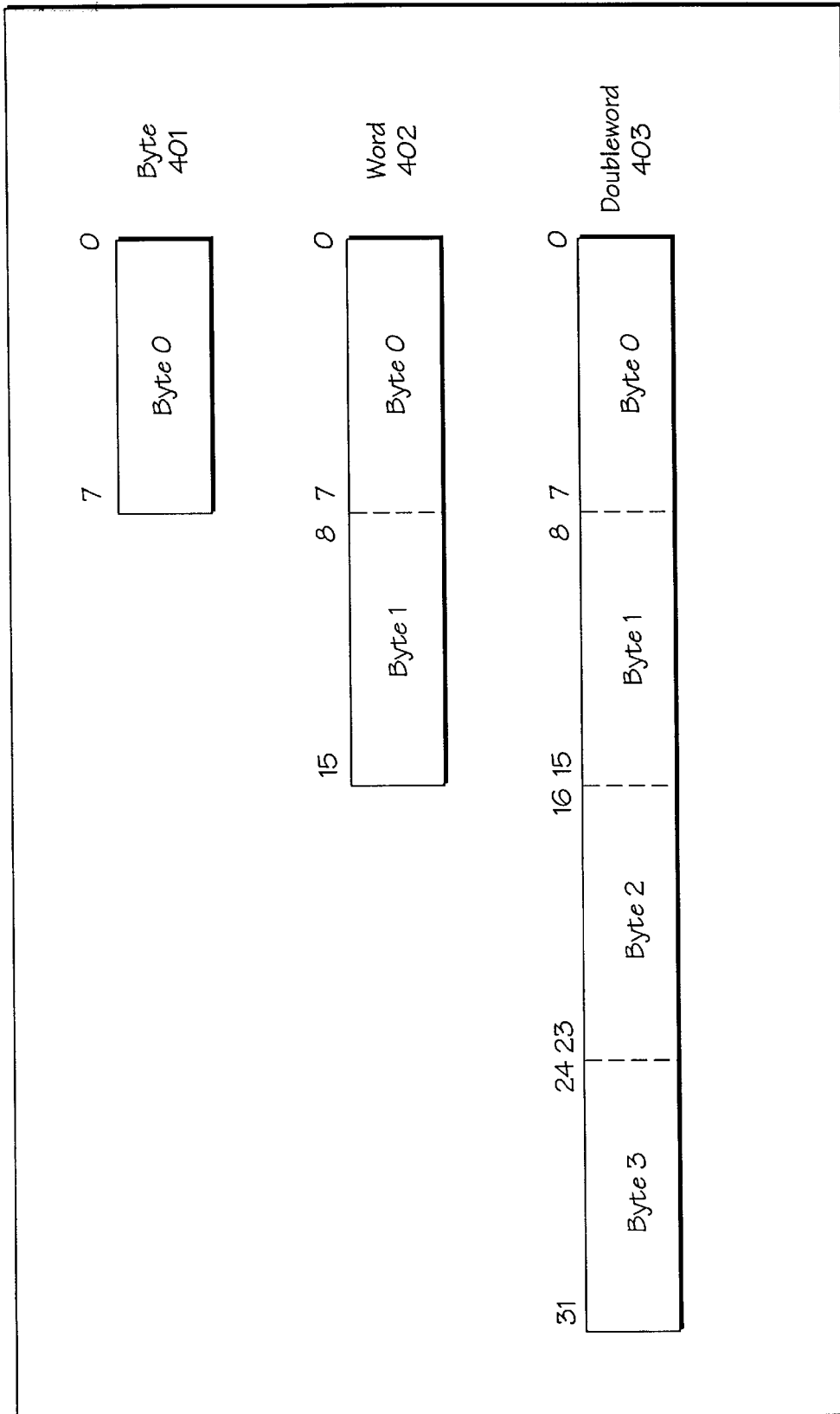


FIG. 4a

U.S. Patent

Mar. 9, 1999

Sheet 5 of 17

5,881,275

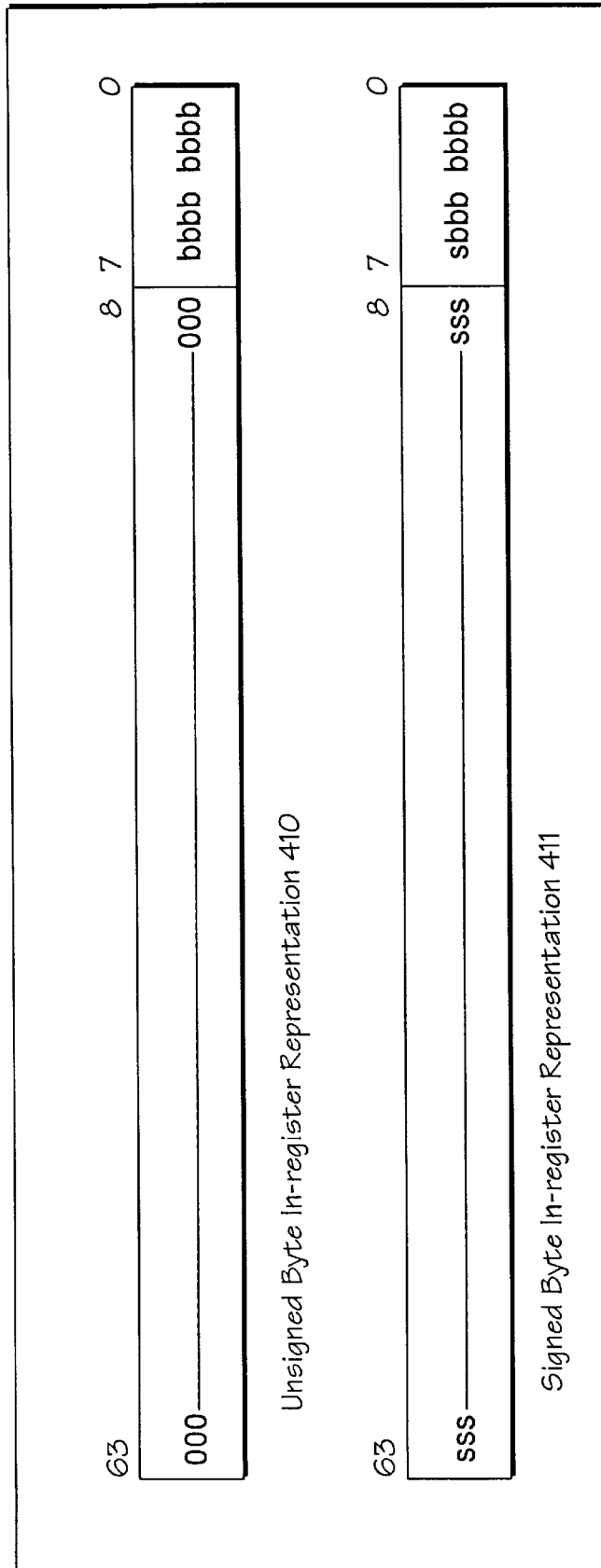


FIG. 4b

U.S. Patent

Mar. 9, 1999

Sheet 6 of 17

5,881,275

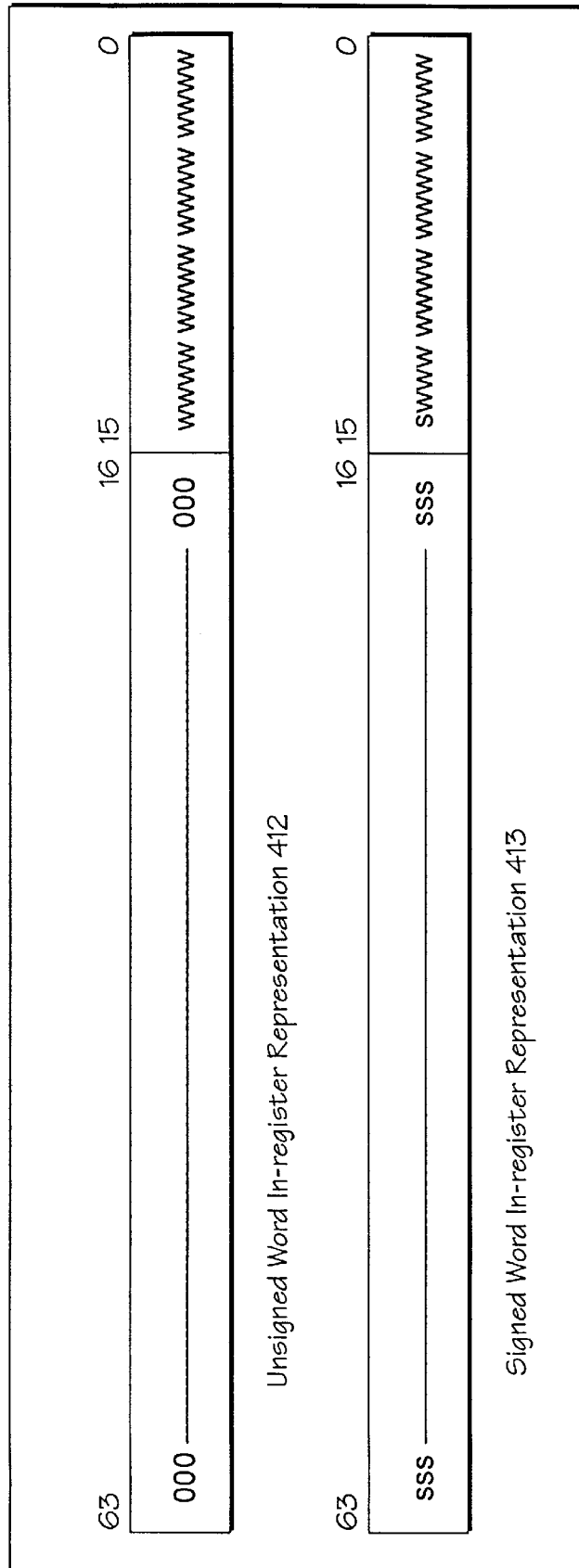


FIG. 4c

U.S. Patent

Mar. 9, 1999

Sheet 7 of 17

5,881,275

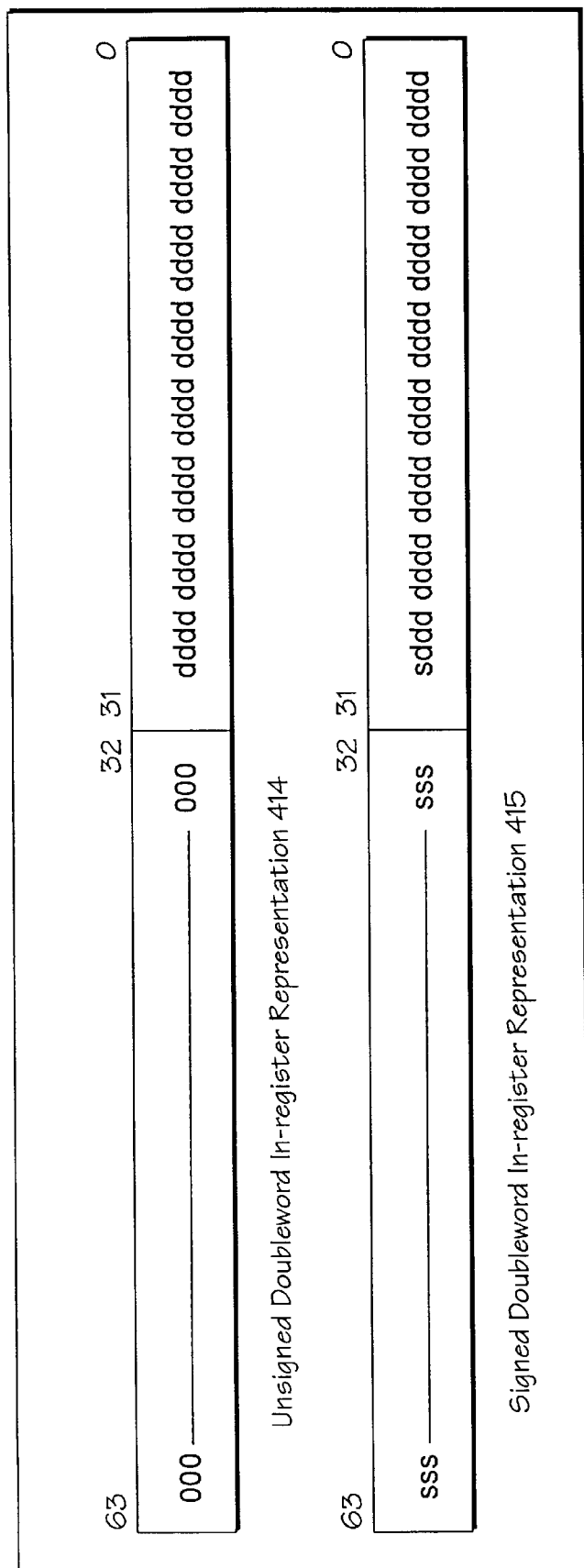


FIG. 4d

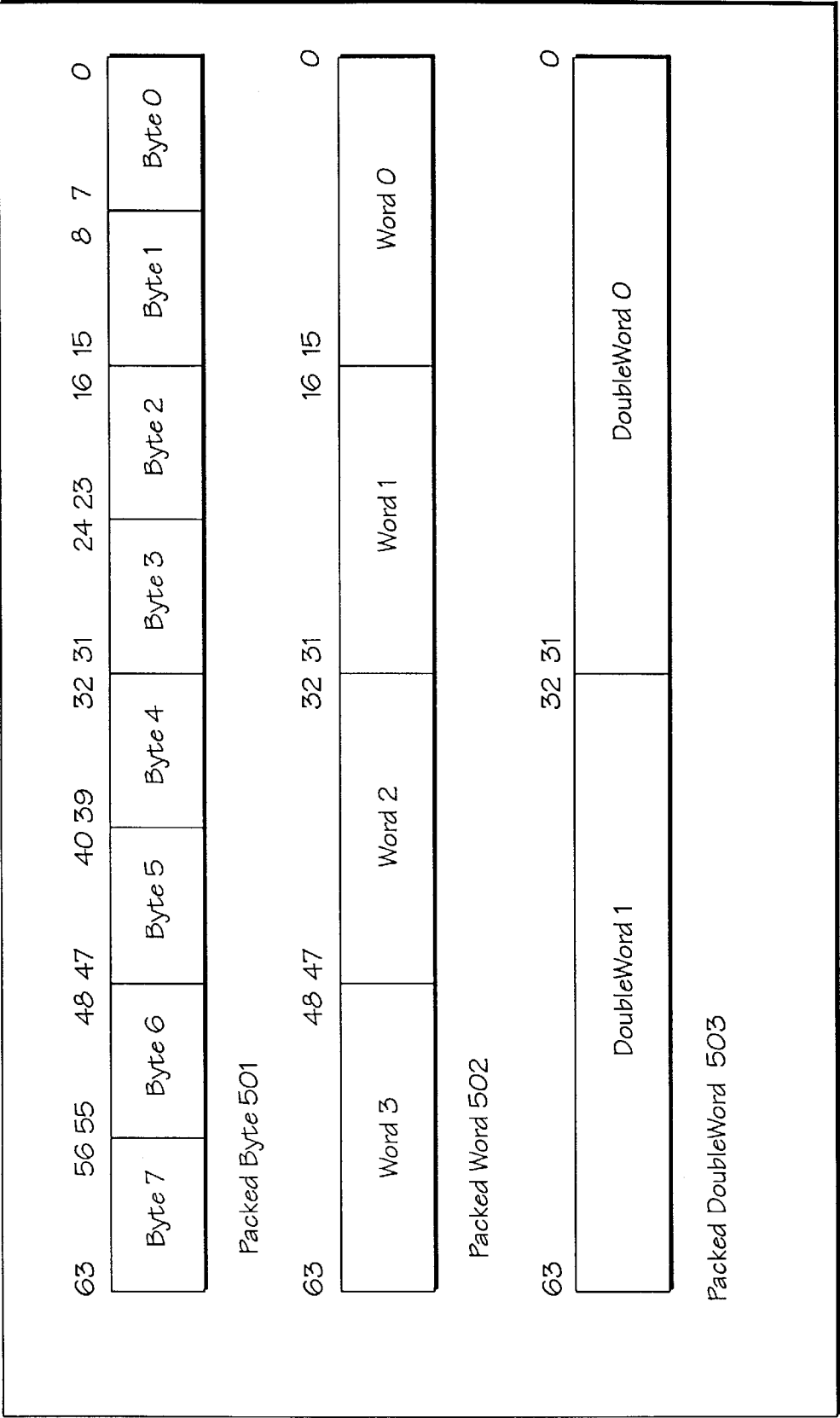


FIG. 5a

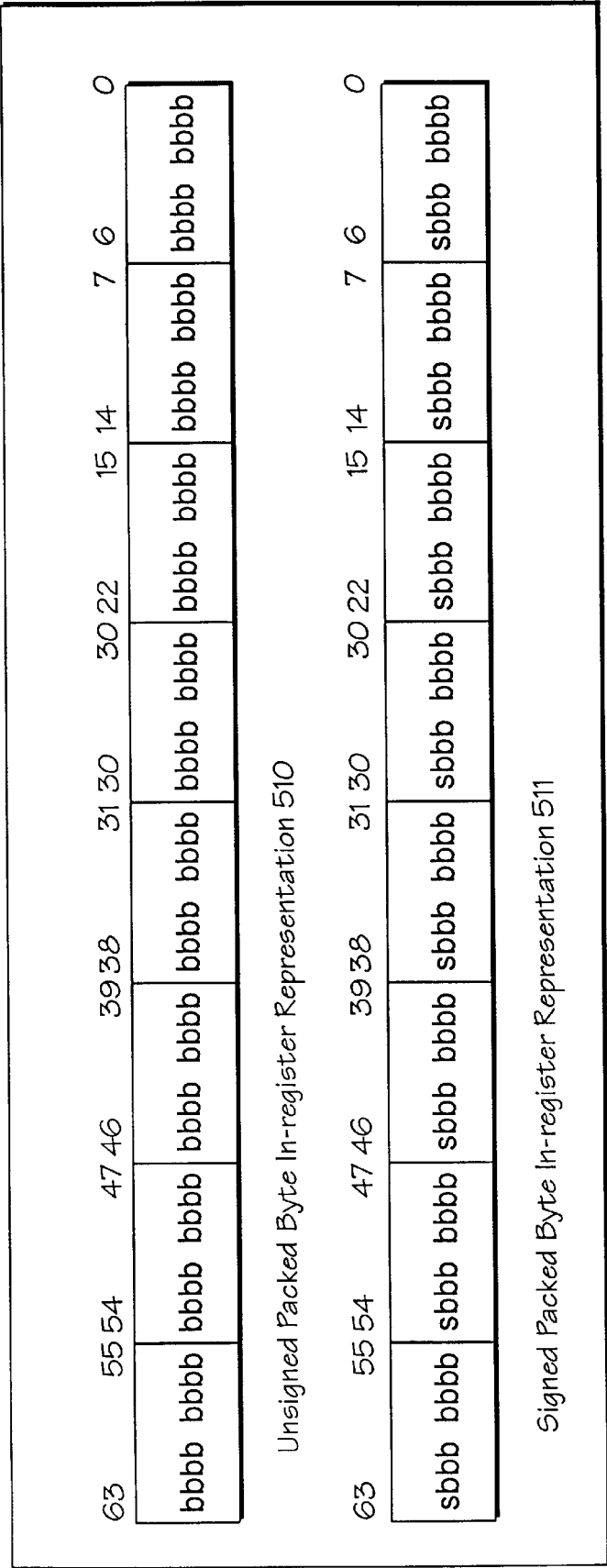


FIG. 5b

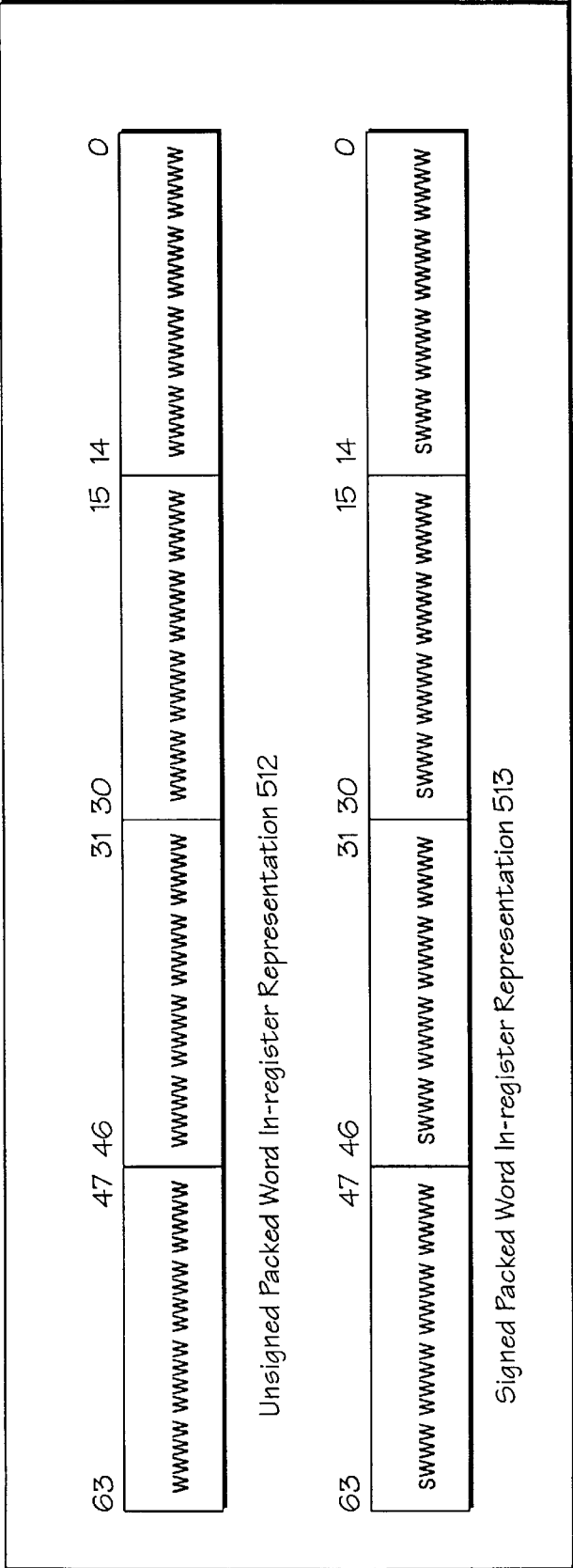


FIG. 5c

U.S. Patent

Mar. 9, 1999

Sheet 11 of 17

5,881,275

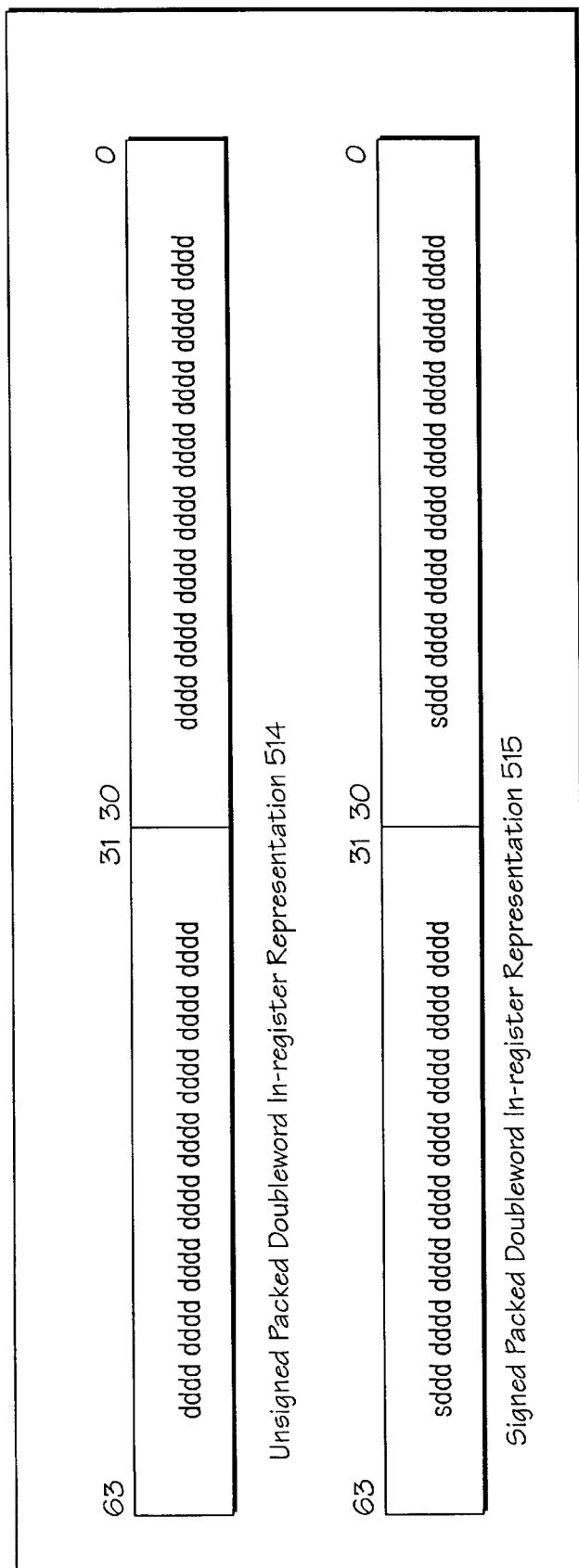


FIG. 5d

U.S. Patent

Mar. 9, 1999

Sheet 12 of 17

5,881,275

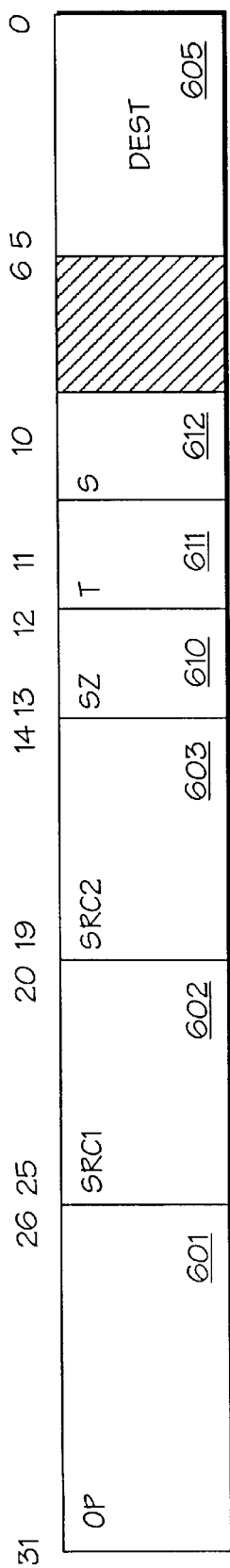


FIG. 6a

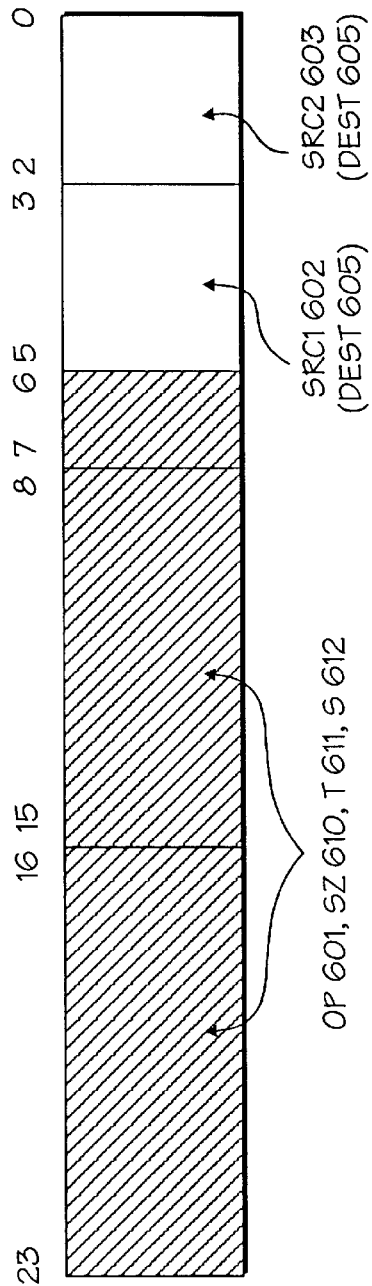


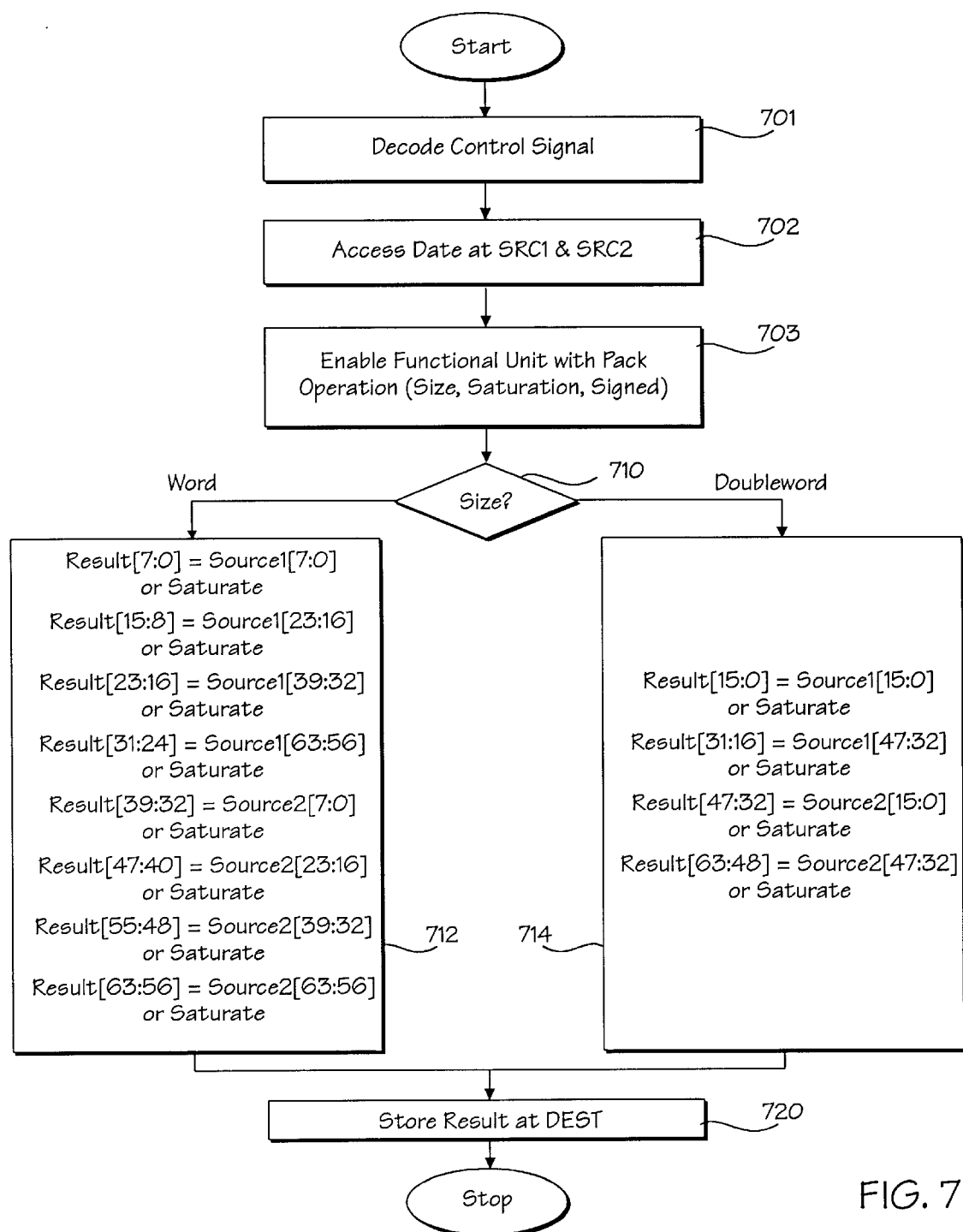
FIG. 6b

U.S. Patent

Mar. 9, 1999

Sheet 13 of 17

5,881,275

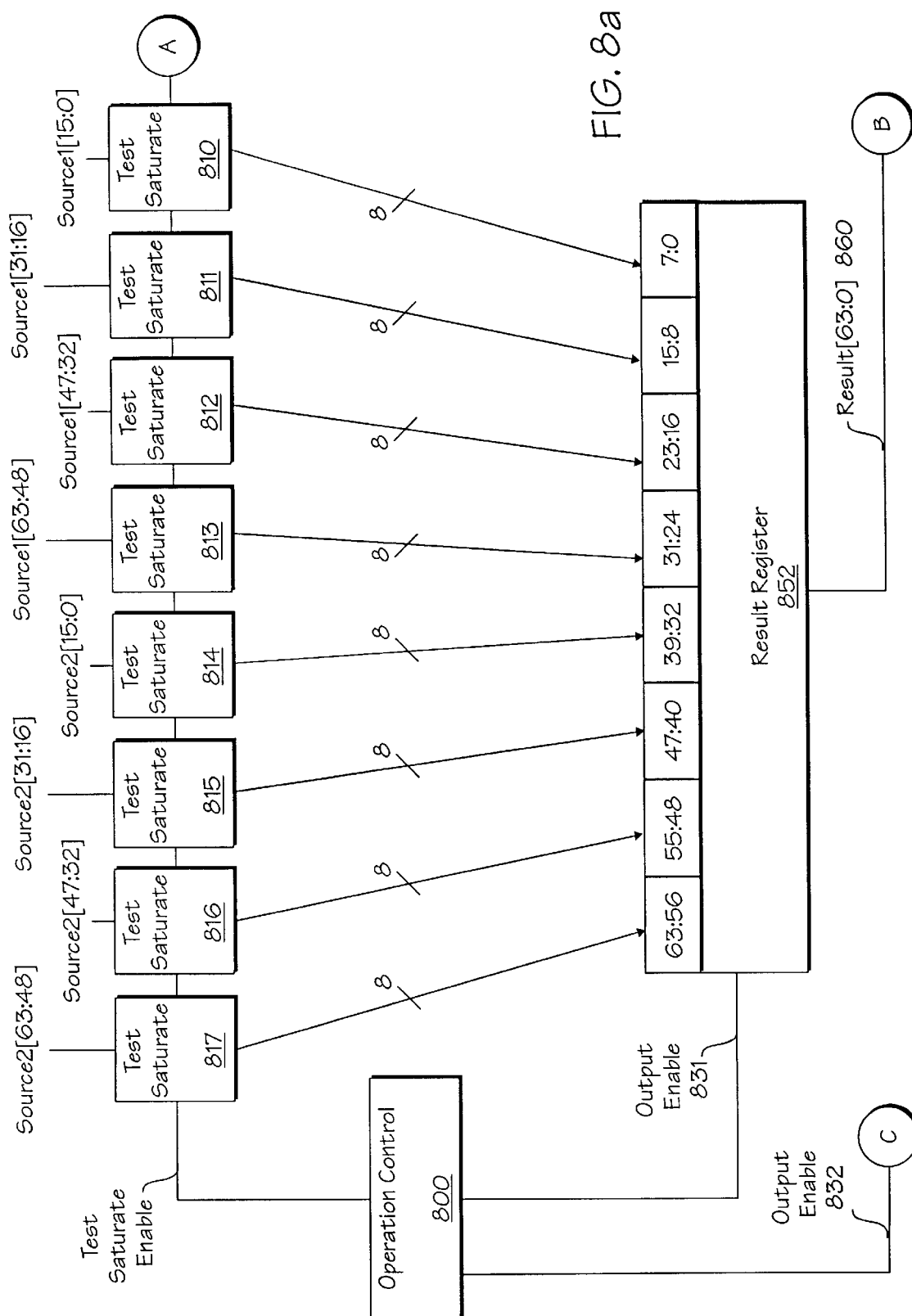


U.S. Patent

Mar. 9, 1999

Sheet 14 of 17

5,881,275

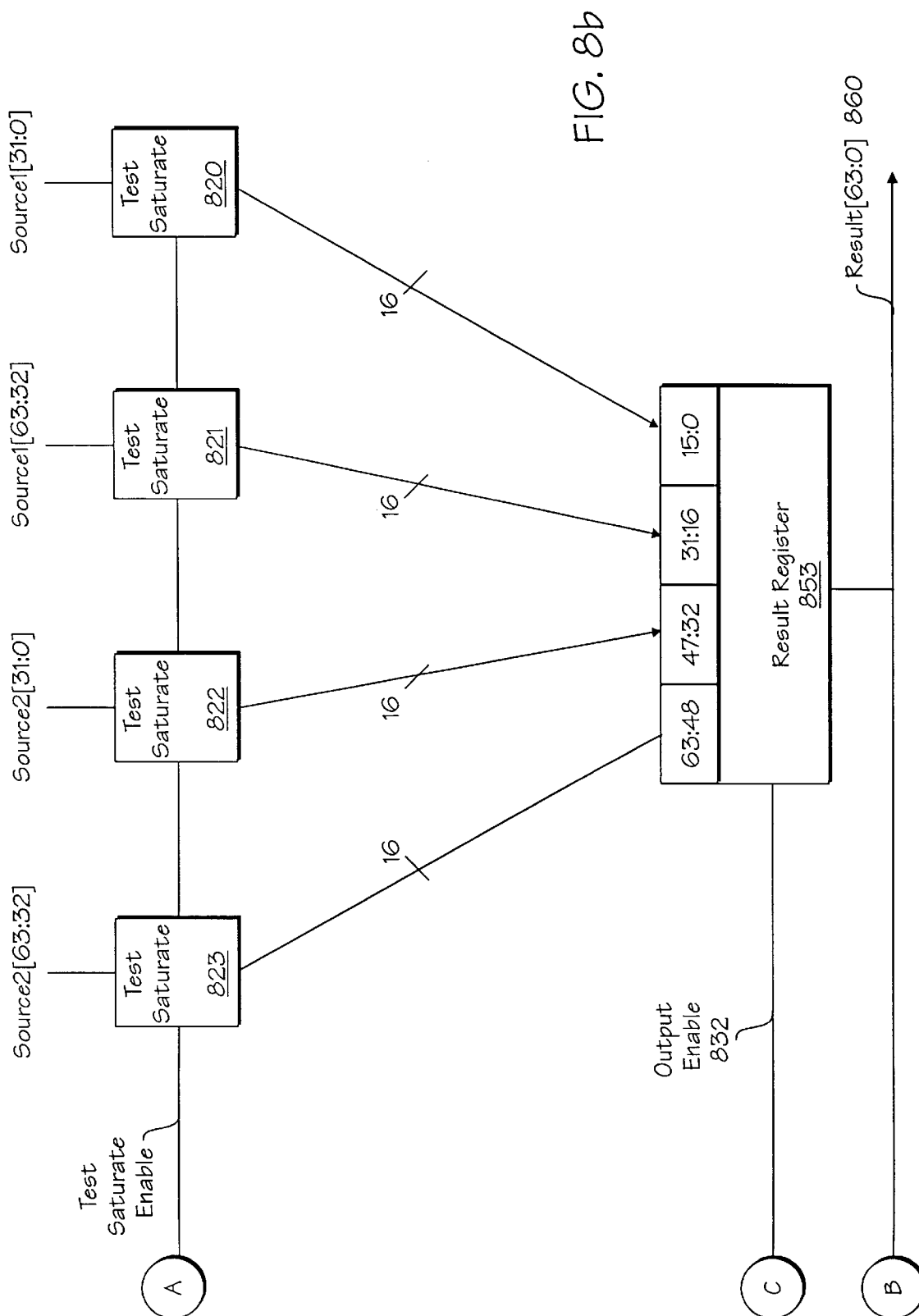


U.S. Patent

Mar. 9, 1999

Sheet 15 of 17

5,881,275

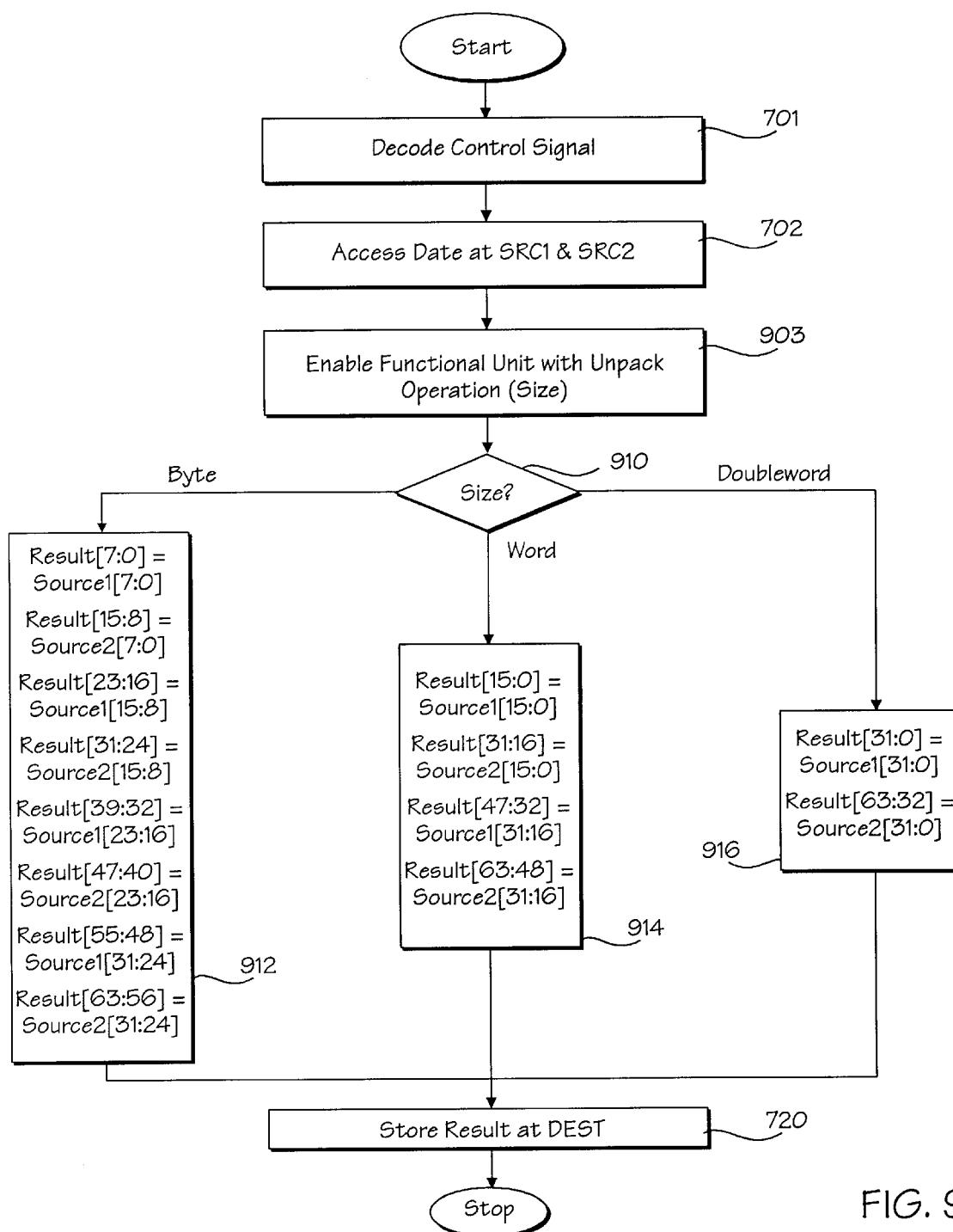


U.S. Patent

Mar. 9, 1999

Sheet 16 of 17

5,881,275

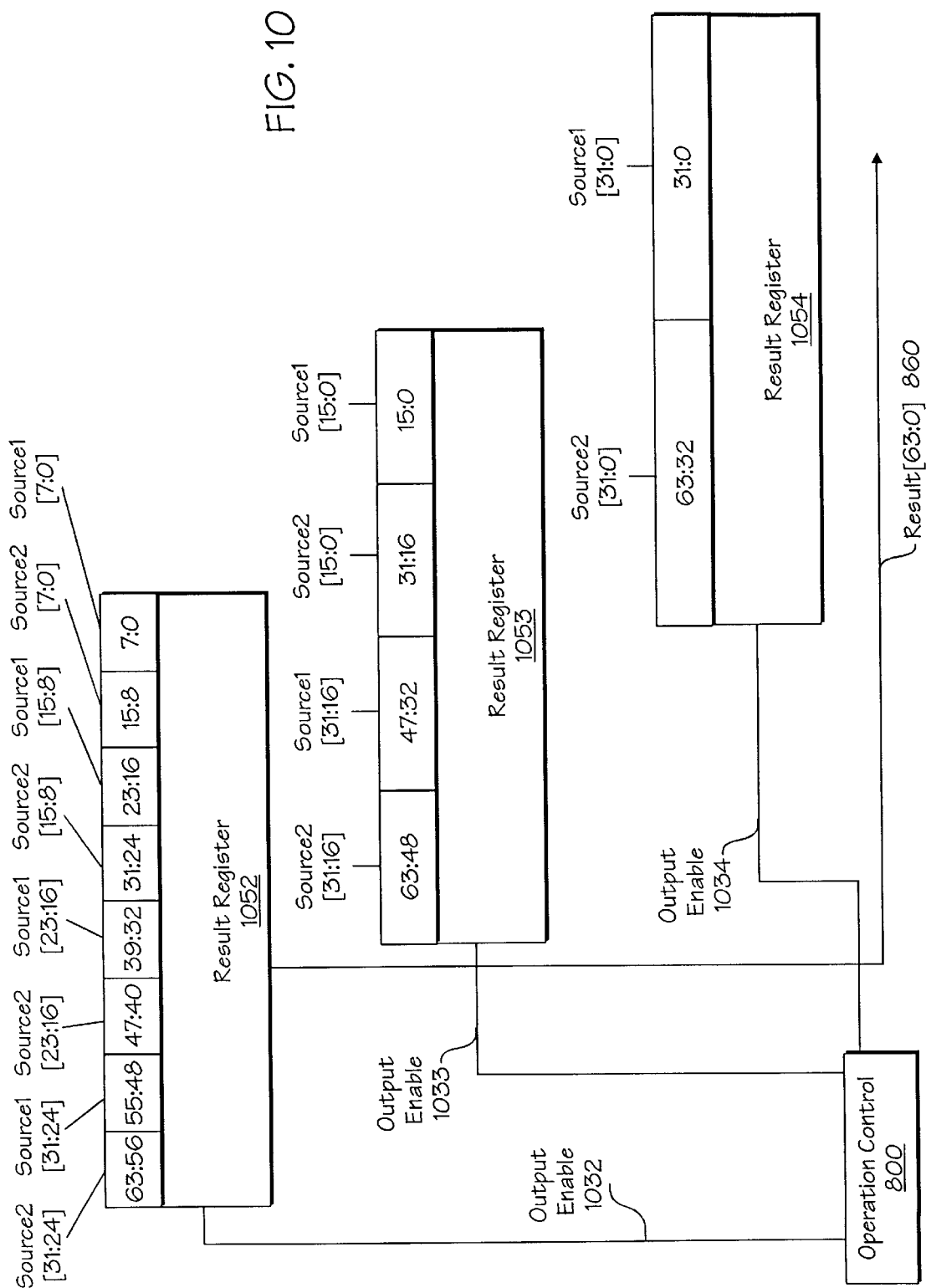


U.S. Patent

Mar. 9, 1999

Sheet 17 of 17

5,881,275



5,881,275

1

METHOD FOR UNPACKING A PLURALITY OF PACKED DATA INTO A RESULT PACKED DATA

This is a continuation of application Ser. No. 08/626,698, 5
filed Apr. 2, 1996, now abandoned, which is a continuation
of application Ser. No. 08/349,047, filed Dec. 2, 1994, now
abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention includes an apparatus and method 10
of performing operations using a single control signal to
manipulate multiple data elements. The present invention
allows execution of move, pack and unpack operations on
packed data types.

2. Description of Related Art

Today, most personal computer systems operate with one 15
instruction to produce one result. Performance increases are
achieved by increasing execution speed of instructions and
the processor instruction complexity, and by performing
multiple instructions in parallel; known as Complex Instruc-
tion Set Computer (CISC). Such processors as the Intel
80386™ microprocessor, available from Intel Corp. of Santa
Clara, Calif., belong to the CISC category of processor.

Previous computer system architecture has been opti- 20
mized to take advantage of the CISC concept. Such systems
typically have data buses thirty-two bits wide. However,
applications targeted at computer supported cooperation
(CSC —the integration of teleconferencing with mixed
media data manipulation), 2D/3D graphics, image
processing, video compression/decompression, recognition
algorithms and audio manipulation increase the need for
improved performance. But, increasing the execution speed
and complexity of instructions is only one solution.

One common aspect of these applications is that they 25
often manipulate large amounts of data where only a few bits
are important. That is, data whose relevant bits are repre-
sented in much fewer bits than the size of the data bus. For
example, processors execute many operations on eight bit
and sixteen bit data (e.g., pixel color components in a video
image) but have much wider data busses and registers. Thus,
a processor having a thirty-two bit data bus and registers,
and executing one of these algorithms, can waste up to
seventy-five percent of its data processing, carrying and
storage capacity because only the first eight bits of data are 30
important.

As such, what is desired is a processor that increases 35
performance by more efficiently using the difference
between the number of bits required to represent the data to
be manipulated and the actual data carrying and storage
capacity of the processor.

SUMMARY OF THE INVENTION

A processor having improved data manipulation opera- 40
tions is described.

A processor. The processor includes a first register for
storing a first packed data, a decoder, and a functional unit.
The decoder has a control signal input. The control signal
input is for receiving a first control signal and a second 45
control signal. The first control signal is for indicating a pack
operation. The second control signal is for indicating an
unpack operation. The functional unit is coupled to the
decoder and the register. The functional unit is for perform-
ing the pack operation and the unpack operation using the
first packed data. The processor also supports a move
operation.

2

Although a great deal of detail has been included in the
description and figures, the invention is defined by the scope
of the claims. Only limitations found in those claims apply
to the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example,
and not limitation, in the figures. Like references indicate
similar elements.

FIG. 1 illustrates an embodiment of the computer system
using the methods and apparatus of the present invention.

FIG. 2 illustrates an embodiment of the processor of the
present invention.

FIG. 3 is a flow diagram illustrating the general steps used
by the processor to manipulate data in the register file.

FIG. 4a illustrates memory data types.

FIG. 4b, FIG. 4c and FIG. 4d illustrate in-register integer
data representations.

FIG. 5a illustrates packed data types.

FIG. 5b, FIG. 5c and FIG. 5d illustrate in-register packed
data representations.

FIG. 6a illustrates a control signal format used in the
computer system to indicate the use of packed data.

FIG. 6b illustrates a second control signal format that can
be used in the computer system to indicate the use of packed
data or integer data.

FIG. 7 illustrates one embodiment of a method followed
by a processor when performing a pack operation on packed
data.

FIG. 8a illustrates a circuit capable of implementing a
pack operation on packed byte data.

FIG. 8b illustrates a circuit capable of implementing a
pack operation on packed word data.

FIG. 9 illustrates an embodiment of a method followed by
a processor when performing an unpack operation on packed
data.

FIG. 10 illustrates a circuit capable of implementing an
unpack operation on packed data.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Overview Of One Embodiment Of The Present Invention

A processor having move, pack, and unpack operations
that operate on multiple data elements is described. In the
following description, numerous specific details are set forth
such as circuits, etc., in order to provide a thorough under-
standing of the present invention. In other instances, well-
known structures and techniques have not been shown in
detail in order not to unnecessarily obscure the present
invention.

Definitions

To provide a foundation for understanding the description
of the embodiments of the present invention, the following
definitions are provided.

Bit X through Bit Y: defines a subfield of binary number.

For example, bit six through bit zero of the byte
00111010₂ (shown in base two) represent the subfield
111010₂. The '2' following a binary number indicates
base 2. Therefore, 1000₂ equals 8₁₀, while F₁₆ equals
15₁₀.

5,881,275

3

R_x: is a register. A register is any device capable of storing and providing data. Further functionality of a register is described below. A register is not necessarily part of the processor's package.

DEST: is a data address.

SRC1: is a data address.

SRC2: is a data address.

Result: is the data to be stored in the register addressed by DEST.

Source1: is the data stored in the register addressed by SRC1.

Source2: is the data stored in the register addressed by SRC2.

Computer System

Referring to FIG. 1, a computer system upon which an embodiment of the present invention can be implemented is shown as computer system 100. Computer system 100 comprises a bus 101, or other communications hardware and software, for communicating information, and a processor 109 coupled with bus 101 for processing information. Computer system 100 further comprises a random access memory (RAM) or other dynamic storage device (referred to as main memory 104), coupled to bus 101 for storing information and instructions to be executed by processor 109. Main memory 104 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 109. Computer system 100 also comprises a read only memory (ROM) 106, and/or other static storage device, coupled to bus 101 for storing static information and instructions for processor 109. Data storage device 107 is coupled to bus 101 for storing information and instructions.

Furthermore, a data storage device 107, such as a magnetic disk or optical disk, and its corresponding disk drive, can be coupled to computer system 100. Computer system 100 can also be coupled via bus 101 to a display device 121 for displaying information to a computer user. Display device 121 can include a frame buffer, specialized graphics rendering devices, a cathode ray tube (CRT), and/or a flat panel display. An alphanumeric input device 122, including alphanumeric and other keys, is typically coupled to bus 101 for communicating information and command selections to processor 109. Another type of user input device is cursor control 123, such as a mouse, a trackball, a pen, a touch screen, or cursor direction keys for communicating direction information and command selections to processor 109, and for controlling cursor movement on display device 121. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), which allows the device to specify positions in a plane. However, this invention should not be limited to input devices with only two degrees of freedom.

Another device which may be coupled to bus 101 is a hard copy device 124 which may be used for printing instructions, data, or other information on a medium such as paper, film, or similar types of media. Additionally, computer system 100 can be coupled to a device for sound recording, and/or playback 125, such as an audio digitizer coupled to a microphone for recording information. Further, the device may include a speaker which is coupled to a digital to analog (D/A) converter for playing back the digitized sounds.

Also, computer system 100 can be a terminal in a computer network (e.g., a LAN). Computer system 100 would

4

then be a computer subsystem of a computer system including a number of networked devices. Computer system 100 optionally includes video digitizing device 126. Video digitizing device 126 can be used to capture video images that can be transmitted to others on the computer network.

Computer system 100 is useful for supporting computer supported cooperation (CSC—the integration of teleconferencing with mixed media data manipulation), 2D/3D graphics, image processing, video compression/decompression, recognition algorithms and audio manipulation.

Processor

FIG. 2 illustrates a detailed diagram of processor 109. Processor 109 can be implemented on one or more substrates using any of a number of process technologies, such as, BiCMOS, CMOS, and NMOS.

Processor 109 comprises a decoder 202 for decoding control signals and data used by processor 109. Data can then be stored in register file 204 via internal bus 205. As a matter of clarity, the registers of an embodiment should not be limited in meaning to a particular type of circuit. Rather, a register of an embodiment need only be capable of storing and providing data, and performing the functions described herein.

Depending on the type of data, the data may be stored in integer registers 201, registers 209, status registers 208, or instruction pointer register 211. Other registers can be included in the register file 204, for example, floating point registers. In one embodiment, integer registers 201 store thirty-two bit integer data. In one embodiment, registers 209 contains eight registers, R₀ 212a through R₇ 212h. Each register in registers 209 is sixty-four bits in length. R₁ 212a, R₂ 212b and R₃ 212c are examples of individual registers in registers 209. Thirty-two bits of a register in registers 209 can be moved into an integer register in integer registers 201. Similarly, an value in an integer register can be moved into thirty-two bits of a register in registers 209.

Status registers 208 indicate the status of processor 109. Instruction pointer register 211 stores the address of the next instruction to be executed. Integer registers 201, registers 209, status registers 208, and instruction pointer register 211 all connect to internal bus 205. Any additional registers would also connect to the internal bus 205.

In another embodiment, some of these registers can be used for two different types of data. For example, registers 209 and integer registers 201 can be combined where each register can store either integer data or packed data. In another embodiment, registers 209 can be used as floating point registers. In this embodiment, packed data can be stored in, registers 209 or floating point data. In one embodiment, the combined registers are sixty-four bits in length and integers are represented as sixty-four bits. In this embodiment, in storing packed data and integer data, the registers do not need to differentiate between the two data types.

Functional unit 203 performs the operations carried out by processor 109. Such operations may include shifts, addition, subtraction and multiplication, etc. Functional unit 203 connects to internal bus 205. Cache 206 is an optional element of processor 109 and can be used to cache data and/or control signals from, for example, main memory 104. Cache 206 is connected to decoder 202, and is connected to receive control signal 207.

FIG. 3 illustrates the general operation of processor 109. That is, FIG. 3 illustrates the steps followed by processor

5,881,275

5

109 while performing an operation on packed data, performing an operation on unpacked data, or performing some other operation. For example, such operations include a load operation to load a register in register file 204 with data from cache 206, main memory 104, read only memory (ROM) 106, or data storage device 107. In one embodiment of the present invention, processor 109 supports most of the instructions supported by the Intel 80486™, available from Intel Corporation of Santa Clara, Calif. In another embodiment of the present invention, processor 109 supports all the operations supported by the Intel 80486™, available from Intel Corporation of Santa Clara, Calif. In another embodiment of the present invention, processor 109 supports all the operations supported by the Pentium™ processor, the Intel 80486™ processor, the 80386™ processor, the Intel 80286™ processor, and the Intel 8086™ processor, all available from Intel Corporation of Santa Clara, Calif. In another embodiment of the present invention, processor 109 supports all the operations supported in the IA™—Intel Architecture, as defined by Intel Corporation of Santa Clara, Calif. (see *Microprocessors*, Intel Data Books volume 1 and volume 2, 1992 and 1993, available from Intel of Santa Clara, Calif.). Generally, processor 109 can support the present instruction set for the Pentium™ processor, but can also be modified to incorporate future instructions, as well as those described herein. What is important is that general processor 109 can support previously used operations in addition to the operations described herein.

At step 301, the decoder 202 receives a control signal 207 from either the cache 206 or bus 101. Decoder 202 decodes the control signal to determine the operations to be performed.

Decoder 202 accesses the register file 204, or a location in memory, at step 302. Registers in the register file 204, or memory locations in the memory, are accessed depending on the register address specified in the control signal 207. For example, for an operation on packed data, control signal 207 can include SRC1, SRC2 and DEST register addresses. SRC1 is the address of the first source register. SRC2 is the address of the second source register. In some cases, the SRC2 address is optional as not all operations require two source addresses. If the SRC2 address is not required for an operation, then only the SRC1 address is used. DEST is the address of the destination register where the result data is stored. In one embodiment, SRC1 or SRC2 is also used as DEST. SRC1, SRC2 and DEST are described more fully in relation to FIG. 6a and FIG. 6b. The data stored in the corresponding registers is referred to as Source1, Source2, and Result respectively. Each of these data is sixty-four bits in length.

In another embodiment of the present invention, any one, or all, of SRC1, SRC2 and DEST, can define a memory location in the addressable memory space of processor 109. For example, SRC1 may identify a memory location in main memory 104 while SRC2 identifies a first register in integer registers 201, and DEST identifies a second register in registers 209. For simplicity of the description herein, references are made to the accesses to the register file 204, however, these accesses could be made to memory instead.

In another embodiment of the present invention, the operation code only includes two addresses, SRC1 and SRC2. In this embodiment, the result of the operation is stored in the SRC1 or SRC2 register. That is SRC1 (or SRC2) is used as the DEST. This type of addressing is compatible with previous CISC instructions having only two addresses. This reduces the complexity in the decoder 202. Note, in this embodiment, if the data contained in the SRC1

6

register is not to be destroyed, then that data must first be copied into another register before the execution of the operation. The copying would require an additional instruction. To simplify the description herein, the three address addressing scheme will be described (i.e. SRC1, SRC2, and DEST). However, it should be remembered that the control signal, in one embodiment, may only include SRC1 and SRC2, and that SRC1 (or SRC2) identifies the destination register.

Where the control signal requires an operation, at step 303, functional unit 203 will be enabled to perform this operation on accessed data from register file 204. Once the operation has been performed in functional unit 203, at step 304, the result is stored back into register file 204 according to requirements of control signal 207.

Data and Storage Formats

FIG. 4a illustrates some of the data formats as may be used in the computer system of FIG. 1. These data formats are fixed point. Processor 109 can manipulate these data formats. Multimedia algorithms often use these data formats. A byte 401 contains eight bits of information. A word 402 contains sixteen bits of information, or two bytes. A doubleword 403 contains thirty-two bits of information, or four bytes. Thus, processor 109 executes control signals that may operate on any one of these memory data formats.

In the following description, references to bit, byte, word, and doubleword subfields are made. For example, bit six through bit zero of the byte 00111010₂ (shown in base 2) represent the subfield 111010₂.

FIG. 4b through FIG. 4d illustrate in-register representations used in one embodiment of the present invention. For example, unsigned byte in-register representation 410 can represent data stored in a register in integer registers 201. In one embodiment, a register, in integer registers 201, is sixty-four bits in length. In another embodiment, a register, in integer registers 201, is thirty-two bits in length. For the simplicity of the description, the following describes sixty-four bit integer registers, however, thirty-two bit integer registers can be used.

Unsigned byte in-register representation 410 illustrates processor 109 storing a byte 401 in integer registers 201, the first eight bits, bit seven through bit zero, in that register are dedicated to the data byte 401. These bits are shown as {b}. To properly represent this byte, the remaining 56 bits must be zero. For an signed byte in-register representation 411, integer registers 201 store the data in the first seven bits, bit six through bit zero, to be data. The seventh bit represents the sign bit, shown as an {s}. The remaining bit sixty-three through bit eight are the continuation of the sign for the byte.

Unsigned word in-register representation 412 is stored in one register of integer registers 201. Bit fifteen through bit zero contain an unsigned word 402. These bits are shown as {w}. To properly represent this word, the remaining bit sixty-three through bit sixteen must be zero. A signed word 402 is stored in bit fourteen through bit zero as shown in the signed word in-register representation 413. The remaining bit sixty-three through bit fifteen is the sign field.

A doubleword 403 can be stored as an unsigned doubleword in-register representation 414 or a signed doubleword in-register representation 415. Bit thirty-one through bit zero of an unsigned doubleword in-register representation 414 are the data. These bits are shown as {d}. To properly represent this unsigned doubleword, the remaining bit sixty-three through bit thirty-two must be zero. Integer registers 201 stores a signed doubleword in-register representation

5,881,275

7

415 in its bit thirty through bit zero; the remaining bit sixty-three through bit thirty-one are the sign field.

As indicated by the above FIG. 4b through FIG. 4d, storage of some data types in a sixty-four bit wide register is an inefficient method of storage. For example, for storage of an unsigned byte in-register representation 410 bit sixty-three through bit eight must be zero, while only bit seven through bit zero may contain non-zero bits. Thus, a processor storing a byte in a sixty-four bit register uses only 12.5% of the register's capacity. Similarly, only the first few bits of operations performed by functional unit 203 will be important.

FIG. 5a illustrates the data formats for packed data. Each packed data includes more than one independent data element. Three packed data formats are illustrated; packed byte 501, packed word 502, and packed doubleword 503. Packed byte, in one embodiment of the present invention, is sixty-four bits long containing eight data elements. Each data element is one byte long. Generally, a data element is an individual piece of data that is stored in a single register (or memory location) with other data elements of the same length. In one embodiment of the present invention, the number of data elements stored in a register is sixty-four bits divided by the length in bits of a data element.

Packed word 502 is sixty-four bits long and contains four word 402 data elements. Each word 402 data element contains sixteen bits of information.

Packed doubleword 503 is sixty-four bits long and contains two doubleword 403 data elements. Each doubleword 403 data element contains thirty-two bits of information.

FIG. 5b through FIG. 5d illustrate the in-register packed data storage representation. Unsigned packed byte in-register representation 510 illustrates the storage of packed byte 501 in one of the registers R₀ 212a through R_n 212af. Information for each byte data element is stored in bit seven through bit zero for byte zero, bit fifteen through bit eight for byte one, bit twenty-three through bit sixteen for byte two, bit thirty-one through bit twenty-four for byte three, bit thirty-nine through bit thirty-two for byte four, bit forty-seven through bit forty for byte five, bit fifty-five through bit forty-eight for byte six and bit sixty-three through bit fifty-six for byte seven. Thus, all available bits are used in the register. This storage arrangement increases the storage efficiency of the processor. As well, with eight data elements accessed, one operation can now be performed on eight data elements simultaneously. Signed packed byte in-register representation 511 is similarly stored in a register in registers 209. Note that only the eighth bit of every byte data element is the necessary sign bit; other bits may or may not be used to indicate sign.

Unsigned packed word in-register representation 512 illustrates how word three through word zero are stored in one register of registers 209. Bit fifteen through bit zero contain the data element information for word zero, bit thirty-one through bit sixteen contain the information for data element word one, bit forty-seven through bit thirty-two contain the information for data element word two and bit sixty-three through bit forty-eight contain the information for data element word three. Signed packed word in-register representation 513 is similar to the unsigned packed word in-register representation 512. Note that only the sixteenth bit of each word data element contains the necessary sign indicator.

Unsigned packed doubleword in-register representation 514 shows how registers 209 store two doubleword data elements. Doubleword zero is stored in bit thirty-one

8

through bit zero of the register. Doubleword one is stored in bit sixty-three through bit thirty-two of the register. Signed packed doubleword in-register representation 515 is similar to unsigned packed doubleword in-register representation 514. Note that the necessary sign bit is the thirty-second bit of the doubleword data element.

As mentioned previously, registers 209 may be used for both packed data and integer data. In this embodiment of the present invention, the individual programming processor 109 may be required to track whether an addressed register, R₁ 212a for example, is storing packed data or simple integer/fixed point data. In an alternative embodiment, processor 109 could track the type of data stored in individual registers of registers 209. This alternative embodiment could then generate errors if, for example, a packed addition operation were attempted on simple/fixed point integer data.

Control Signal Formats

The following describes one embodiment of control signal formats used by processor 109 to manipulate packed data. In one embodiment of the present invention, control signals are represented as thirty-two bits. Decoder 202 may receive control signal 207 from bus 101. In another embodiment, decoder 202 can also receive such control signals from cache 206.

FIG. 6a illustrates a general format for a control signal operating on packed data. Operation field OP 601, bit thirty-one through bit twenty-six, provides information about the operation to be performed by processor 109; for example, packed addition, packed subtraction, etc. SRC1 602, bit twenty-five through twenty, provides the source register address of a register in registers 209. This source register contains the first packed data, Source1, to be used in the execution of the control signal. Similarly, SRC2 603, bit nineteen through bit fourteen, contains the address of a register in registers 209. This second source register contains the packed data, Source2, to be used during execution of the operation. DEST 605, bit five through bit zero, contains the address of a register in registers 209. This destination register will store the result packed data, Result, of the packed data operation.

Control bits SZ 610, bit twelve and bit thirteen, indicates the length of the data elements in the first and second packed data source registers. If SZ 610 equals 01₂, then the packed data is formatted as packed byte 501. If SZ 610 equals 10₂, then the packed data is formatted as packed word 502. SZ 610 equaling 00₂ or 11₂ is reserved, however, in another embodiment, one of these values could be used to indicate packed doubleword 503.

Control bit T 611, bit eleven, indicates whether the operation is to be carried out with saturate mode. If T 611 equals one, then a saturating operation is performed. If T 611 equals zero, then a nonsaturating operation is performed. Saturating operations will be described later.

Control bit S 612, bit ten, indicates the use of a signed operation. If S 612 equals one, then a signed operation is performed. If S 612 equals zero, then an unsigned operation is performed. FIG. 6b illustrates a second general format for a control signal operating on packed data. This format corresponds with the general integer opcode format described in the "Pentium™ Processor Family User's Manual," available from Intel Corporation, Literature Sales, P.O. Box 7641, Mt. prospect, Ill., 60056-7641. Note that OP 601, SZ 610, T 611, and S 612 are all combined into one large field. For some control signals, bits three through five are SRC1 602. In one embodiment, where there is a SRC1

5,881,275

9

602 address, then bits three through five also correspond to DEST 605. In an alternate embodiment, where there is a SRC2 603 address, then bits zero through two also correspond to DEST 605. For other control signals, like a packed shift immediate operation, bits three through five represent an extension to the opcode field. In one embodiment, this extension allows a programmer to include an immediate value with the control signal, such as a shift count value. In one embodiment, the immediate value follows the control signal. This is described in more detail in the "Pentium™ Processor Family User's Manual," in appendix F, pages F-1 through F-3. Bits zero through two represent SRC2 603. This general format allows register to register, memory to register, register by memory, register by register, register by immediate, register to memory addressing. Also, in one embodiment, this general format can support integer register to register, and register to integer register addressing.

Description of Saturate/Unsaturate

As mentioned previously, T 611 indicates whether operations optionally saturate. Where the result of an operation, with saturate enabled, overflows or underflows the range of the data, the result will be clamped. Clamping means setting the result to a maximum or minimum value should a result exceed the range's maximum or minimum value. In the case of underflow, saturation clamps the result to the lowest value in the range and in the case of overflow, to the highest value. The allowable range for each data format is shown in Table 1.

TABLE 1

| Data Format | Minimum Value | Maximum Value |
|---------------------|------------------|--------------------|
| Unsigned Byte | 0 | 255 |
| Signed Byte | -128 | 127 |
| Unsigned Word | 0 | 65535 |
| Signed Word | -32768 | 32767 |
| Unsigned Doubleword | 0 | 2 ⁶⁴ -1 |
| Signed Doubleword | -2 ⁶³ | 2 ⁶³ -1 |

As mentioned above, T 611 indicates whether saturating operations are being performed. Therefore, using the unsigned byte data format, if an operation's result=258 and saturation was enabled, then the result would be clamped to 255 before being stored into the operation's destination register. Similarly, if an operation's result=-32999 and processor 109 used signed word data format with saturation enabled, then the result would be clamped to -32768 before being stored into the operation's destination register.

Data Manipulation Operations

In one embodiment of the present invention, the performance of multimedia applications is improved by not only supporting a standard CISC instruction set (unpacked data operations), but by supporting operations on packed data. Such packed data operations can include an addition, a subtraction, a multiplication, a compare, a shift, an AND, and an XOR. However, to take full advantage of these operations, it has been determined that data manipulation operations should be included. Such data manipulation operations can include a move, a pack, and an unpack. Move, pack and unpack facilitate the execution of the other operations by generating packed data in formats that allow for easier use by programmers.

For further background on the other packed operations, see "A Microprocessor Having a Compare Operation," filed

10

on Dec. 2, 1994, Ser. No. 08/349,040, now abandoned, "A Microprocessor Having a Multiply Operation," filed on Dec. 1, 1994, Ser. No. 08/349,559, now abandoned, "A Novel Processor Having Shift Operations," filed on Dec. 1, 1994, Ser. No. 08/349,730, now abandoned, "A Method and Apparatus Using Packed Data in a Processor," filed on Dec. 30, 1993, Ser. No. 08/176,123, now abandoned and "A Method and Apparatus Using Novel Operations in a Processor," filed on Dec. 30, 1993, Ser. No. 08/175,772, now abandoned all assigned to the assignee of the present invention.

Move Operation

The move operation transfers data to or from registers 209. In one embodiment, SRC2 603 is the address containing the source data and DEST 605 is the address where the data is to be transferred. In this embodiment, SRC1 602 would not be used. In another embodiment, SRC1 602 is DEST 605.

For the purposes of the explanation of the move operation, a distinction is drawn between a register and a memory location. Registers are found in register file 204 while memory can be, for example, in cache 206, main memory 104, ROM 106, data storage device 107.

The move operation can move data from memory to registers 209, from registers 209 to memory, and from a register in registers 209 to a second register in registers 209. In one embodiment, packed data is stored in different registers than those used to store integer data. In this embodiment, the move operation can move data from integer registers 201 to registers 209. For example, in processor 109, if packed data is stored in registers 209 and integer data is stored in integer registers 201, then a move instruction can be used to move data from integer registers 201 to registers 209, and vice versa.

In one embodiment, when a memory address is indicated for the move, the eight bytes of data at the memory location (the memory location indicating the least significant byte) are loaded to a register in registers 209 or stored from that register. When a register in registers 209 is indicated, the contents of that register are moved to or loaded from a second register in registers 209. If the integer registers 201 are sixty-four bits in length, and an integer register is specified, then the eight bytes of data in that integer register are loaded to a register in registers 209 or stored from that register.

In one embodiment, integers are represented as thirty-two bits. When a move operation is performed from registers 209 to integer registers 201, then only the low thirty-two bits of the packed data are moved to the specified integer register. In one embodiment, the high order thirty-two bits are zeroed. Similarly, only the low thirty-two bits of a register in registers 209 are loaded when a move is executed from integer registers 201 to registers 209. In one embodiment, processor 109 supports a thirty-two bit move operation between a register in registers 209 and memory. In another embodiment, a move of only thirty-two bits is performed on the high order thirty-two bits of packed data.

Pack Operation

In one embodiment of the present invention, the SRC1 602 register contains data (Source1), the SRC2 603 register contains the data (Source2), and DEST 605 register will contain the result data (Result) of the operation. That is, parts of Source1 and parts of Source2 will be packed together to generate Result.

In one embodiment, a pack operation converts packed words (or doublewords) into packed bytes (or words) by

5,881,275

11

packing the low order bytes (or words) of the source packed words (or doublewords) into the bytes (or words) of the Result. In one embodiment, the pack operation converts quad packed words into packed doublewords. This operation can be optionally performed with signed data. Further, this operation can be optionally performed with saturate.

FIG. 7 illustrates one embodiment of a method of performing a pack operation on packed data. This embodiment can be implemented in the processor 109 of FIG. 2.

At step 701, decoder 202 decodes control signal 207 received by processor 109. Thus, decoder 202 decodes: the operation code for the appropriate pack operation; SRC1 602, SRC2 603 and DEST 605 addresses in registers 209; saturate/unsaturate, signed/unsigned, and length of the data elements in the packed data. As mentioned previously, SRC1 602 (or SRC2 603) can be used as DEST 605.

At step 702, via internal bus 205, decoder 202 accesses registers 209 in register file 204 given the SRC1 602 and SRC2 603 addresses. Registers 209 provides functional unit 203 with the packed data stored in the SRC1 602 register (Source1), and the packed data stored in SRC2 603 register (Source2). That is, registers 209 communicate the packed data to functional unit 203 via internal bus 205.

At step 703, decoder 202 enables functional unit 203 to perform the appropriate pack operation. Decoder 202 further communicates, via internal bus 205, saturate and the size of the data elements in Source1 and Source2. Saturate is optionally used to maximize the value of the data in the result data element. If the value of the data elements in Source1 or Source2 are greater than or less than the range of values that the data elements of Result can represent, then the corresponding result data element is set to its highest or lowest value. For example, if signed values in the word data elements of Source1 and Source2 are smaller than 0x80 (or 0x8000 for doublewords), then the result byte (or word) data elements are clamped to 0x80 (or 0x8000 for doublewords). If signed values in word data elements of Source1 and Source 2 are greater than 0x7F (or 0x7FFF for doublewords), then the result byte (or word) data elements are clamped to 0x7F (or 0x7FFF).

At step 710, the size of the data element determines which step is to be executed next. If the size of the data elements is sixteen bits (packed word 502 data), then functional unit 203 performs step 712. However, if the size of the data elements in the packed data is thirty-two bits (packed doubleword 503 data), then functional unit 203 performs step 714.

12

performed. Source1 bits seven through zero are Result bits seven through zero. Source1 bits twenty-three through sixteen are Result bits fifteen through eight. Source1 bits thirty-nine through thirty-two are Result bits twenty-three through sixteen. Source1 bits sixty-three through fifty-six are Result bits thirty-one through twenty-four. Source2 bits seven through zero are Result bits thirty-nine through thirty-two. Source2 bits twenty-three through sixteen are Result bits forty-seven through forty. Source2 bits thirty-nine through thirty-two are Result bits fifty-five through forty-eight. Source2 bits sixty-three through fifty-six are Result bits thirty-one through twenty-four. If saturate is set, then the high order bits of each word are tested to determine whether the Result data element should be clamped.

Assuming the size of the source data elements is thirty-two bits, then step 714 is executed. In step 714, the following is performed. Source1 bits fifteen through zero are Result bits fifteen through zero. Source1 bits forty-seven through thirty-two are Result bits thirty-one through sixteen. Source2 bits fifteen through zero are Result bits forty-seven through thirty-two. Source2 bits forty-seven through thirty-two are Result bits sixty-three through forty-eight. If saturate is set, then the high order bits of each doubleword are tested to determine whether the Result data element should be clamped.

In one embodiment, the packing of step 712 is performed simultaneously. However, in another embodiment, this packing is performed serially. In another embodiment, some of the packing is performed simultaneously and some is performed serially. This discussion also applies to the packing of step 714.

At step 720, the Result is stored in the DEST 605 register.

Table 2 illustrates the in-register representation of a pack unsigned word operation with no saturation. The first row of bits is the packed data representation of Source1. The second row of bits is the data representation of Source2. The third row of bits is the packed data representation of the Result. The number below each data element bit is the data element number. For example, Source1 data element three is 10000000₂.

TABLE 2

| Source 1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| | 3 | | 2 | | 1 | | 0 |
| Source 2 | | | | | | | |
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| | 3 | | 2 | | 1 | | 0 |
| Result | | | | | | | |
| 00000000 | 00000000 | 00000000 | 10001000 | 01010101 | 11111111 | 01110000 | 10001000 |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | | | | | | | 0 |

Assuming the size of the source data elements is sixteen bits, then step 712 is executed. In step 712, the following is

Table 3 illustrates the in-register representation of pack signed doubleword operation with saturation.

5,881,275

13

14

TABLE 3

| Source 1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| 1 | | | | 0 | | | |
| Source 2 | | | | | | | |
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| 1 | | | | 0 | | | |
| Result | | | | | | | |
| 11000000 | 00000000 | 10000000 | 00000000 | 01111111 | 11111111 | 10000000 | 10000000 |
| 3 | | 2 | | 1 | | 0 | |

Pack Circuits

In one embodiment of the present invention, to achieve efficient execution of the pack operation parallelism is used. FIG. 8a and 8b illustrate one embodiment of a circuit that can perform a pack operation on packed data. The circuit can optionally perform the pack operation with saturation.

The circuit of FIG. 8a and 8b includes an operation control circuit 800, a result register 852, a result register 853, eight sixteen bit to eight bit test saturate circuits, and four thirty-two bit to sixteen bit test saturate circuits.

Operation control 800 receives information from the decoder 202 to enable a pack operation. Operation control 800 uses the saturate value to enable the saturation tests for each of the test saturate circuits. If the size of the source packed data is word packed data 503, then output enable 831 is set by operation control 800. This enables the output of output register 852. If the size of the source packed data is doubleword packed data 504, then output enable 832 is set by operation control 800. This enables the output of output register 853.

Each test saturate circuit can selectively test for saturation. If a test for saturation is disabled, then each test saturate circuit merely passes the low order bits through to a corresponding position in a result register. If a test for saturate is enabled, then each test saturate circuit tests the high order bits to determine if the result should be clamped.

Test saturate 810 through test saturate 817 have sixteen bit inputs and eight bit outputs. The eight bit outputs are the lower eight bits of the inputs, or optionally, are a clamped value (0x80, 0x7F, or 0xFF). Test saturate 810 receives Source1 bits fifteen through zero and outputs bits seven through zero for result register 852. Test saturate 811

852. Test saturate 813 receives Source1 bits sixty-three through forty-eight and outputs bits thirty-one through twenty-four for result register 852. Test saturate 814 receives Source2 bits fifteen through zero and outputs bits thirty-nine through thirty-two for result register 852. Test saturate 815 receives Source2 bits thirty-one through sixteen and outputs bits forty-seven through forty for result register 852. Test saturate 816 receives Source2 bits forty-seven through thirty-two and outputs bits fifty-five through forty-eight for result register 852. Test saturate 817 receives Source2 bits sixty-three through forty-eight and outputs bits sixty-three through fifty-six for result register 852.

Test saturate 820 through test saturate 823 have thirty-two bit inputs and sixteen bit outputs. The sixteen bit outputs are the lower sixteen bits of the inputs, or optionally, are a clamped value (0x8000, 0x7FFF, or 0xFFFF). Test saturate 820 receives Source1 bits thirty-one through zero and outputs bits fifteen through zero for result register 853. Test saturate 821 receives Source1 bits sixty-three through thirty-two and outputs bits thirty-one through sixteen for result register 853. Test saturate 822 receives Source2 bits thirty-one through zero and outputs bits forty-seven through thirty-two for result register 853. Test saturate 823 receives Source2 bits sixty-three through thirty-two and outputs bits sixty-three through forty-eight of result register 853.

For example, in Table 4, a pack word unsigned with no saturate is performed. Operation control 800 will enable result register 852 to output result[63:0] 860.

TABLE 4

| Source 1 | | | | | | | | | | | |
|----------|--|------|------|----------|-------------------|----------|----------|----------|----------|----------|--|
| | | | | | 00001110 01110000 | | 00001110 | | 00001000 | | |
| 3 | | | 2 | | 1 | | 0 | | | | |
| Source 2 | | | | | | | | | | | |
| | | | | | 00001110 10000001 | | 00001110 | | 10000001 | | |
| 3 | | | 2 | | 1 | | 0 | | | | |
| Result | | | | | | | | | | | |
| | | | | 10000001 | | 10000001 | | | | | |
| 7 | | 6 | | 5 | | 4 | | 3 | | 2 | |
| | | | | | | | | 01110000 | | 00001000 | |
| | | | | | | | | 1 | | 0 | |

60

receives Source1 bits thirty-one through sixteen and outputs bits fifteen through eight for result register 852. Test saturate 812 receives Source1 bits forty-seven through thirty-two and outputs bits twenty-three through sixteen for result register

However, if a pack doubleword unsigned with no saturate is performed, operation control 800 will enable result register 853 to output result[63:0] 860. Table 5 illustrates this result.

5,881,275

15

16

TABLE 5

| Source 1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|--|--|
| | | 10001110 | 01000001 | 00001000 | 10001000 | | |
| | | 1 | | | 0 | | |
| Source 2 | | | | | | | |
| | | 10000001 | 00000001 | 00001110 | 10001000 | | |
| | | 1 | | | 0 | | |
| Result | | | | | | | |
| ... | 00001110 | 10000001 | | 00001110 | 00001000 | | |
| 3 | | 2 | 1 | | 0 | | |

Unpack Operation

In one embodiment, an unpack operation interleaves the low order packed bytes, words or doublewords of two source packed data to generate result packed bytes, words, or doublewords.

FIG. 9 illustrates one embodiment of a method of performing an unpack operation on packed data. This embodiment can be implemented in the processor 109 of FIG. 2.

Step 701 and step 702 are executed first. At step 903, decoder 202 enables functional unit 203 to perform the unpack operation. Decoder 202 communicates, via internal bus 205, the size of the data elements in Source1 and Source2.

At step 910, the size of the data element determines which step is to be executed next. If the size of the data elements is eight bits (packed byte 501 data), then functional unit 203 performs step 712. However, if the size of the data elements in the packed data is sixteen bits (packed word 502 data), then functional unit 203 performs step 714. However, if the size of the data elements in the packed data is thirty-two bits (packed doubled word 503 data), then functional unit 203 performs step 716.

Assuming the size of the source data elements is eight bits, then step 712 is executed. In step 712, the following is performed. Source1 bits seven through zero are Result bits seven through zero. Source2 bits seven through zero are Result bits fifteen through eight. Source1 bits fifteen through eight are Result bits twenty-three through sixteen. Source2 bits fifteen through eight are Result bits thirty-one through

twenty-three through sixteen are Result bits forty-seven through forty. Source1 bits thirty-one through twenty-four are Result bits fifty-five through forty-eight. Source2 bits thirty-one through twenty-four are Result bits sixty-three through fifty-six.

Assuming the size of the source data elements is sixteen bits, then step 714 is executed. In step 714, the following is performed. Source1 bits fifteen through zero are Result bits fifteen through zero. Source2 bits fifteen through zero are Result bits thirty-one through sixteen. Source1 bits thirty-one through sixteen are Result bits forty-seven through thirty-two. Source2 bits thirty-one through sixteen are Result bits sixty-three through forty-eight.

Assuming the size of the source data elements is thirty-two bits, then step 716 is executed. In step 716, the following is performed. Source1 bits thirty-one through zero are Result bits thirty-one through zero. Source2 bits thirty-one through zero are Result bits sixty-three through thirty-two.

In one embodiment, the unpacking of step 712 is performed simultaneously. However, in another embodiment, this unpacking is performed serially. In another embodiment, some of the unpacking is performed simultaneously and some is performed serially. This discussion also applies to the unpacking of step 714 and step 716.

At step 720, the Result is stored in the DEST 605 register.

Table 6 illustrates the in-register representation of an unpack byte operation.

TABLE 6

| Source 1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Source 2 | | | | | | | |
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Result | | | | | | | |
| 11110011 | 10000000 | 00000000 | 01110000 | 10001110 | 10001111 | 10001000 | 10001000 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

twenty-four. Source1 bits twenty-three through sixteen are Result bits thirty-nine through thirty-two. Source2 bits

Table 7 illustrates the in-register representation of an unpack word operation.

TABLE 7

| Source 1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| | 3 | | 2 | | 1 | | 0 |

5,881,275

17

18

TABLE 7-continued

| Source 2 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| 3 | | 2 | | 1 | | | 0 |
| Result | | | | | | | |
| 11110011 | 00000000 | 10000000 | 01110000 | 10001000 | 10001111 | 10001111 | 10001000 |
| 3 | | 2 | | 1 | | | 0 |

10

Table 8 illustrates the in-register representation of an unpack doubleword operation.

1052. Source1 bits twenty-three through sixteen are bits thirty-nine through thirty-two for result register **1052**.

TABLE 8

| Source 1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 00101010 | 01010101 | 01010101 | 11111111 | 10000000 | 01110000 | 10001111 | 10001000 |
| 1 | | | | | | | 0 |
| Source 2 | | | | | | | |
| 00000000 | 00000000 | 11000000 | 00000000 | 11110011 | 00000000 | 10001110 | 10001000 |
| 1 | | | | | | | 0 |
| Result | | | | | | | |
| 11110011 | 00000000 | 10001110 | 10001000 | 10000000 | 01110000 | 10001111 | 10001000 |
| 1 | | | | | | | 0 |

Unpack Circuits

In one embodiment of the present invention, to achieve efficient execution of the unpack operation parallelism is used. FIG. 10 illustrates one embodiment of a circuit that can perform an unpack operation on packed data.

The circuit of FIG. 10 includes the operation control circuit **800**, a result register **1052**, a result register **1053**, and a result register **1054**.

Operation control **800** receives information from the decoder **202** to enable an unpack operation. If the size of the source packed data is byte packed data **502**, then output enable **1032** is set by operation control **800**. This enables the output of result register **1052**. If the size of the source packed data is word packed data **503**, then output enable **1033** is set by operation control **800**. This enables the output of output register **1053**. If the size of the source packed data is doubleword packed data **504**, then output enable **1034** is set by operation control **800**. This enables the output of output result register **1054**.

Result register **1052** has the following inputs. Source1 bits seven through zero are bits seven through zero for result

Source2 bits twenty-three through sixteen are bits forty-seven through forty for result register **1052**. Source1 bits thirty-one through twenty-four are bits fifty-five through forty-eight for result register **1052**. Source2 bits thirty-one through twenty-four are bits sixty-three through fifty-six for result register **1052**.

Result register **1053** has the following inputs. Source1 bits fifteen through zero are bits fifteen through zero for result register **1053**. Source2 bits fifteen through zero are bits thirty-one through sixteen for result register **1053**. Source1 bits thirty-one through sixteen are bits forty-seven through thirty-two for result register **1053**. Source2 bits thirty-one through sixteen are bits sixty-three through forty-eight of result register **853**.

Result register **1054** has the following inputs. Source1 bits thirty-one through zero are bits thirty-one through zero for result register **1054**. Source2 bits thirty-one through zero are bits sixty-three through thirty-two of result register **1054**.

For example, in Table 9, an unpack word operation is performed. Operation control **800** will enable result register **1053** to output result[63:0] **860**.

TABLE 9

| Source 1 | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| | | 00001110 | 01110000 | 00001110 | 00001000 | | |
| 3 | 2 | 1 | | | 0 | | |
| Source 2 | | | | | | | |
| | | 00001110 | 00000001 | 00001110 | 10000001 | | |
| 3 | 2 | 1 | | | 0 | | |
| Result | | | | | | | |
| 00001110 | 00000001 | 00001110 | 01110000 | 00001110 | 10000001 | 00001110 | 00001000 |
| 3 | 2 | 1 | | | | | 0 |

register **1052**. Source2 bits seven through zero are bits fifteen through eight for result register **1052**. Source1 bits fifteen through eight are bits twenty-three through sixteen for result register **1052**. Source 2 bits fifteen through eight are bits thirty-one through twenty-four for result register

However, if an unpack doubleword is performed, operation control **800** will enable result register **1054** to output result[63:0] **860**. Table 10 illustrates this result.

5,881,275

19

20

TABLE 10

| Source 1 | | | |
|----------|----------|----------|--|
| | 00001110 | 01000001 | 00001110 00001000 |
| 1 | | | 0 |
| Source 2 | | | |
| | 00001110 | 00000001 | 00001110 10000001 |
| 1 | | | 0 |
| Result | | | |
| 00001110 | 00000001 | 00001110 | 10000001 00001110 01000001 00001110 10001000 |
| 1 | | | 0 |

Therefore, the move, pack, and unpack operations can manipulate multiple data elements. In prior art processors, to perform these types of manipulations, multiple separate operations would be needed to perform a single packed move, pack or unpacked operation. The data lines for the packed data operations, in one embodiment, all carry relevant data. This leads to higher performance computer system.

What is claimed is:

1. A method for manipulating packed data in a computer system comprising the computer implemented steps of:

- a) decoding a Single Instruction Multiple Data (SIMD) unpack instruction, the instruction identifying a first and second packed data respectively including a first plurality of data elements and a second plurality of data elements, each data element consisting of a separate multiple bit data field, each data element in the first plurality of data elements corresponds to a data element in the second plurality of data elements in a respective position; and
- b) simultaneously copying, in response to the unpack instruction, less than all data elements from the first plurality of data elements and corresponding data elements from the second plurality of data elements into a third packed data as a plurality of separate result data elements.

2. The method of claim 1, wherein the step of simultaneously copying includes simultaneously copying half of the data elements in the first plurality of data elements and half the data elements of the second plurality of data elements.

3. The method of claim 2, wherein the first plurality of data elements and the second plurality of data elements each includes either two, four, or eight data elements.

4. The method of claim 1, wherein the step of copying includes interleaving the corresponding data elements from the first and second plurality of data elements into the third packed data as separate result data elements.

5. The method of claim 4, wherein the first plurality of data are copied in the same order as the first plurality of data elements appear in the first packed data sequence.

6. A computer implemented method for manipulating data elements in a first and second packed data in response to a Single Instruction Multiple Data (SIMD) unpack instruction, the first and second packed data respectively including a first plurality of data elements and a second plurality of data elements, each data element consisting of a separate multiple bit data field, wherein each data element in the first plurality of data elements corresponds to a different element in the second plurality of data elements in a respective position, the method comprising the computer implemented steps of:

- a) decoding the SIMD unpack instruction;
- b) reading the first packed data and reading the second packed data;
- c) simultaneously copying, in response to the unpack instruction, less than all data elements from the first plurality of data elements and corresponding data elements from the second plurality of data elements into a third packed data as a third plurality of separate data elements.

7. The method of claim 6, wherein the step of simultaneously copying includes simultaneously copying half of the data elements in the first plurality of data elements and half the data elements of the second plurality of data elements.

8. The method of claim 7, wherein the first plurality of data elements and the second plurality of data elements each includes either two, four, or eight data elements.

9. The method of claim 6, wherein the step of copying includes interleaving the corresponding data elements from the first and second plurality of data elements into the third packed data as separate result data elements.

10. The method of claim 9, wherein the first plurality of data are copied in the same order as the first plurality of data elements appear in the first packed data sequence.

* * * * *

EXHIBIT 6



US006385634B1

(12) **United States Patent**
Peleg et al.

(10) **Patent No.:** **US 6,385,634 B1**
(45) **Date of Patent:** ***May 7, 2002**

(54) **METHOD FOR PERFORMING MULTIPLY-ADD OPERATIONS ON PACKED DATA**

(75) Inventors: **Alexander D. Peleg**, Carmelia (IL);
Millind Mittal, South San Francisco;
Larry M. Mennemeier, Boulder Creek,
both of CA (US); **Benny Eitan**, Haifa
(IL); **Carole Dulong**, Saratoga, CA
(US); **Eiichi Kowashi**, Ryugasaki (JP);
Wolf Witt, Sunnyvale, CA (US)

(73) Assignee: **Intel Corporation**, Santa Clara, CA
(US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

OTHER PUBLICATIONS

J. Shipnes, *Graphics Processing with the 88110 RISC Microprocessor*, IEEE (1992), pp. 169–174.

MC88110 Second Generation RISC Microprocessor User's Manual, Motorola Inc. (1991).

Errata to MC88110 Second Generation RISC Microprocessor User's Manual, Motorola Inc. (1992), pp. 1–11.

MC88110 Programmer's Reference Guide, Motorola Inc. (1992), p. 1–4.

i860™ Microprocessor Family Programmer's Reference Manual, Intel Corporation (1992), Ch. 1, 3, 8, 12.

R. B. Lee, *Accelerating Multimedia With Enhanced Microprocessors*, IEEE Micro (Apr. 1995), pp 22–32.

TMS320C2x User's Guide, Texas Instruments (1993) pp 3–2 through 3–11; 3–28 through 3–34; 4–1 through 4–22; 4–41; 4–103; 4–119 through 4–120; 4–122; 4–150 through 4–151.

L. Gwennap, *New PA-RISC Processor Decodes MPEG Video*, Microprocessor Report (Jan. 1994), pp 16, 17.

SPARC Technology Business, *UltraSPARC Multimedia Capabilities On-Chip Support for Real-Time Video Advanced Graphics*, Sun Microsystems (Sep. 1994).

(List continued on next page.)

(21) Appl. No.: **08/522,067**

(22) Filed: **Aug. 31, 1995**

(51) **Int. Cl.**⁷ **G06F 7/38**

(52) **U.S. Cl.** **708/490; 708/523; 708/524;**
708/603; 708/626; 712/221

(58) **Field of Search** 364/736, 754,
364/750.5, 758; 395/650, 675, 750, 800;
708/490, 620, 523, 524, 603, 626, 685,
706; 712/36, 42, 221, 222

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,711,692 A 1/1973 Batcher 708/210
3,723,715 A 3/1973 Chen et al. 708/709

(List continued on next page.)

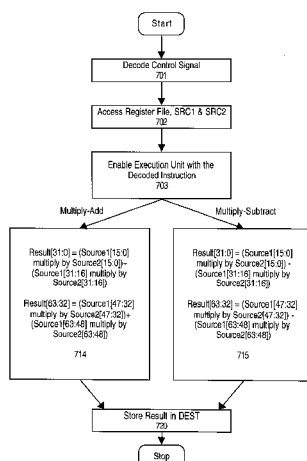
Primary Examiner—Emmanuel L. Moise

(74) *Attorney, Agent, or Firm*—Blakely, Sokoloff, Taylor & Zafman LLP

(57) **ABSTRACT**

A method and apparatus for including in a processor instructions for performing multiply-add operations on packed data. In one embodiment, a processor is coupled to a memory. The memory has stored therein a first packed data and a second packed data. The processor performs operations on data elements in said first packed data and said second packed data to generate a third packed data in response to receiving an instruction. At least two of the data elements in this third packed data storing the result of performing multiply-add operations on data elements in the first and second packed data.

14 Claims, 7 Drawing Sheets



US 6,385,634 B1

Page 2

U.S. PATENT DOCUMENTS

| | | | | | | | |
|---------------|---------|----------------------|---------|-------------|---------|-----------------------|---------|
| 4,161,784 A | 7/1979 | Cushing et al. | 708/513 | 5,487,022 A | 1/1996 | Simpson et al. | 708/205 |
| 4,344,151 A | 8/1982 | White | 708/622 | 5,506,865 A | 4/1996 | Weaver, Jr. | 370/335 |
| 4,393,468 A | 7/1983 | New | 708/518 | 5,509,129 A | 4/1996 | Guttag et al. | 712/203 |
| 4,418,383 A | 11/1983 | Doyle et al. | 710/127 | 5,517,438 A | 5/1996 | Dao-Trong et al. | 708/501 |
| 4,498,177 A | 2/1985 | Larson | 714/806 | 5,528,529 A | 6/1996 | Seal | 708/628 |
| 4,707,800 A | 11/1987 | Montrone et al. | 708/714 | 5,576,983 A | 11/1996 | Shiokawa | 708/622 |
| 4,771,379 A | 9/1988 | Ando et al. | 712/42 | | | | |
| 4,989,168 A | 1/1991 | Kuroda et al. | 708/210 | | | | |
| 5,095,457 A | 3/1992 | Jeong | 708/626 | | | | |
| 5,111,422 A * | 5/1992 | Ulrich | 708/603 | | | | |
| 5,187,679 A | 2/1993 | Vassiliadis | 708/706 | | | | |
| 5,222,037 A | 6/1993 | Taniguchi | 708/497 | | | | |
| 5,227,994 A | 7/1993 | Ohki Mitsuharu | 708/603 | | | | |
| 5,241,492 A | 8/1993 | Girardeau, Jr. | 708/523 | | | | |
| 5,262,976 A | 11/1993 | Young et al. | 708/628 | | | | |
| 5,293,558 A | 3/1994 | Narita et al. | 708/605 | | | | |
| 5,321,644 A | 6/1994 | Schibinger | 708/530 | | | | |
| 5,325,320 A | 6/1994 | Chiu | 708/629 | | | | |
| 5,420,815 A | 5/1995 | Nix et al. | 708/603 | | | | |
| 5,442,799 A * | 8/1995 | Murakami et al. | 712/36 | | | | |
| 5,457,805 A * | 10/1995 | Nakamura | 708/603 | | | | |

OTHER PUBLICATIONS

Y. Kawakami et al., *LSI Applications: A Single-Chip Digital Processor for Voiceband Applications*, Solid State Circuits Conference, Digest of Technical Papers; IEEE International (1980).

B. Case, *Philips Hopes to Displace DSPs with VLIW*, Microprocessor Report (Dec. 94), pp 12–18.

N. Margulis, *i860 Microprocessor Architecture*, McGraw Hill, Inc. (1990) Ch. 6, 7, 8, 10, 11.

Pentium Processor User's Manual, vol. 3: Architecture and Programming Manual, Intel Corporation (1993), Ch. 1, 3, 4, 6, 8, and 18.

* cited by examiner

U.S. Patent

May 7, 2002

Sheet 1 of 7

US 6,385,634 B1

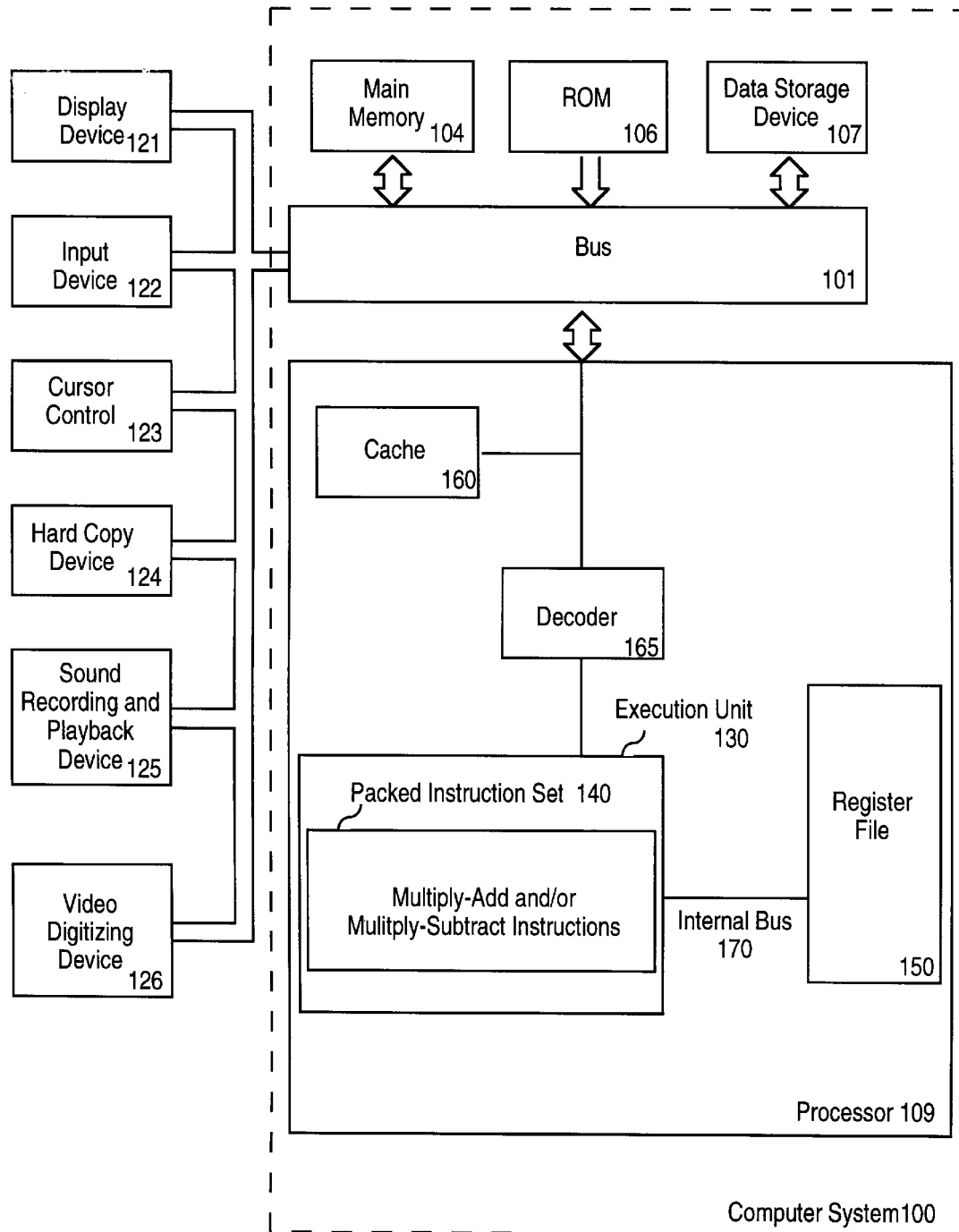


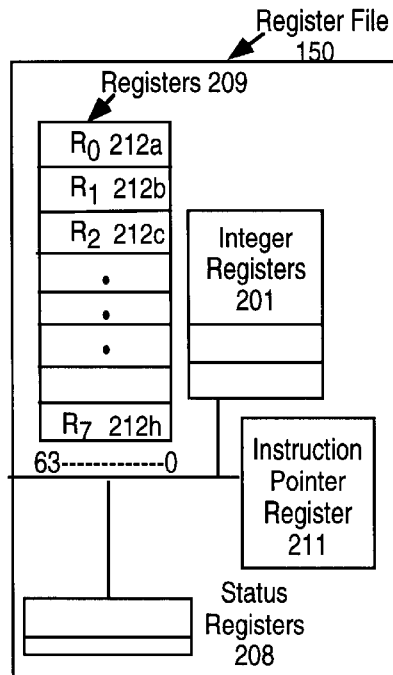
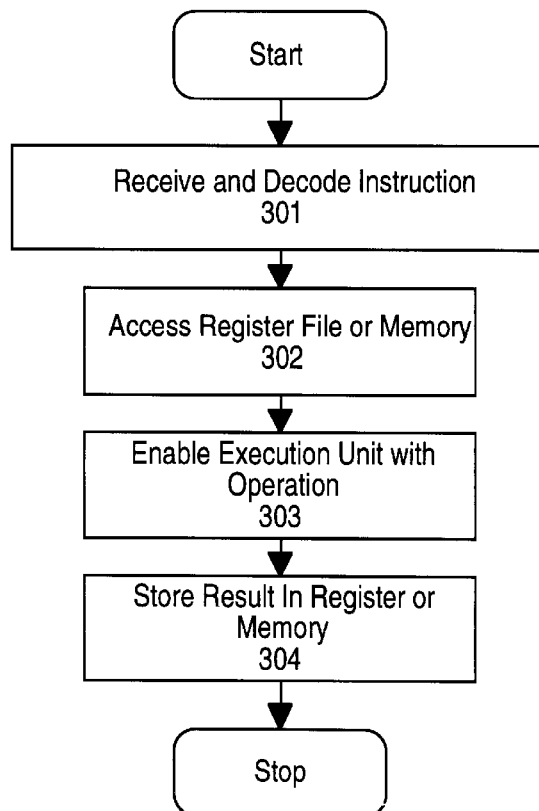
Figure 1

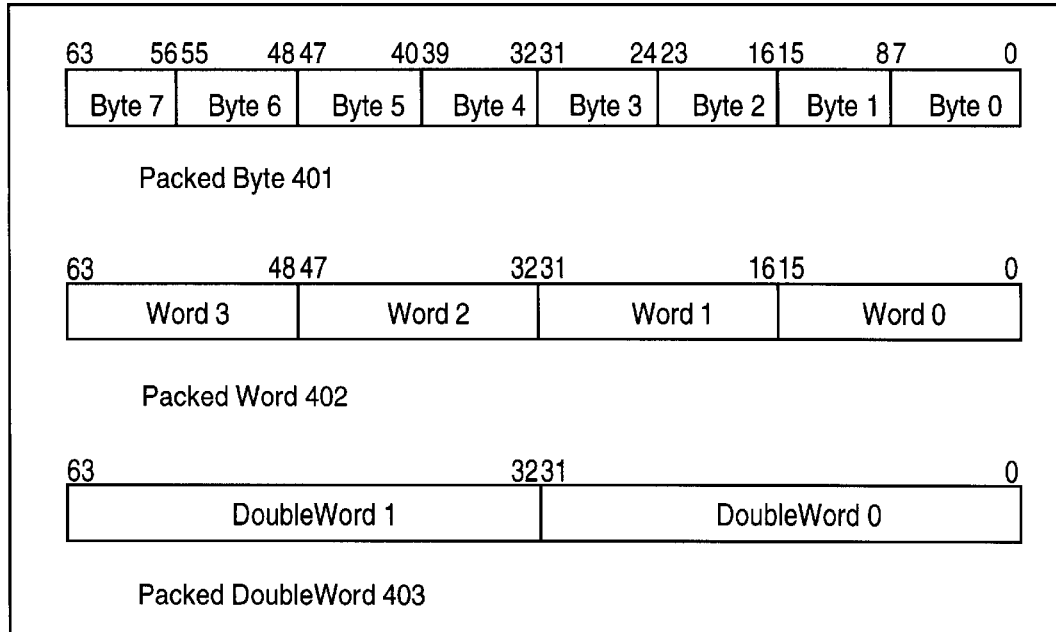
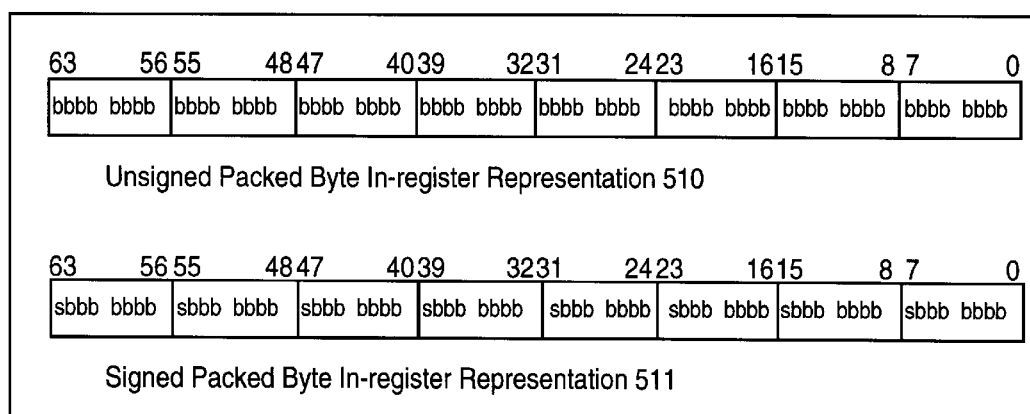
U.S. Patent

May 7, 2002

Sheet 2 of 7

US 6,385,634 B1

**Figure 2****Figure 3**

U.S. Patent**May 7, 2002****Sheet 3 of 7****US 6,385,634 B1****Figure 4****Figure 5a**

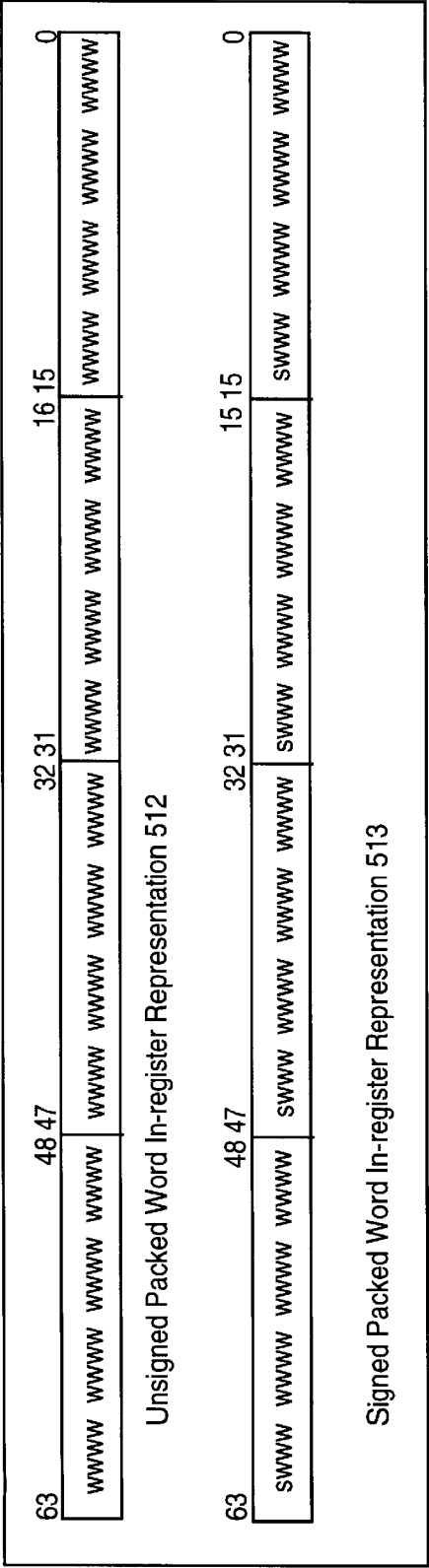


Figure 5b

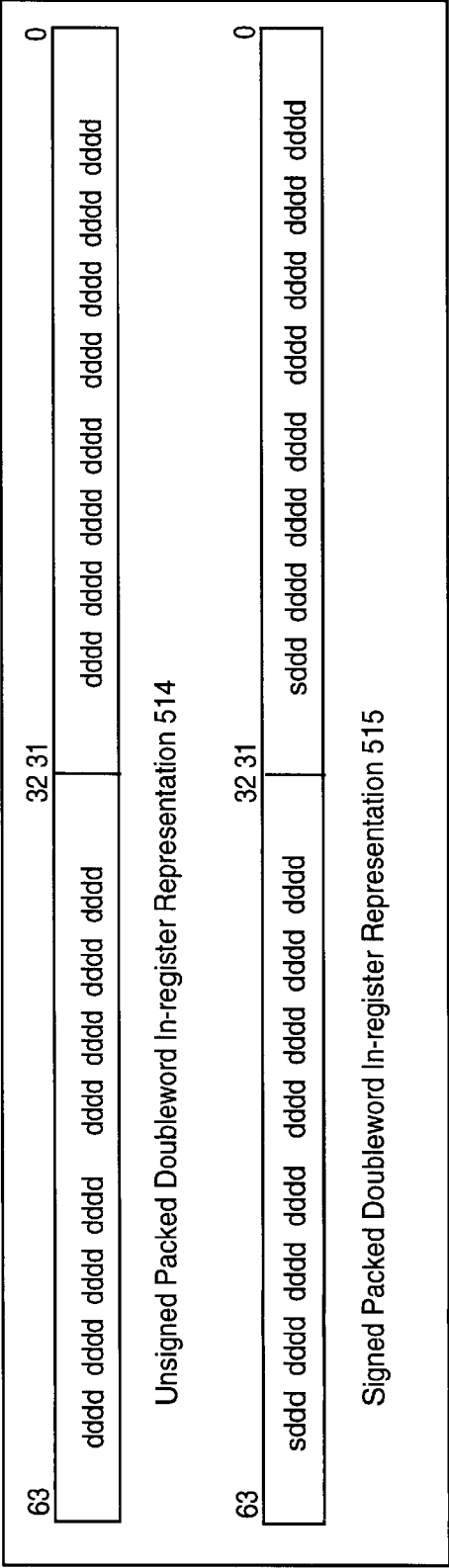


Figure 5c

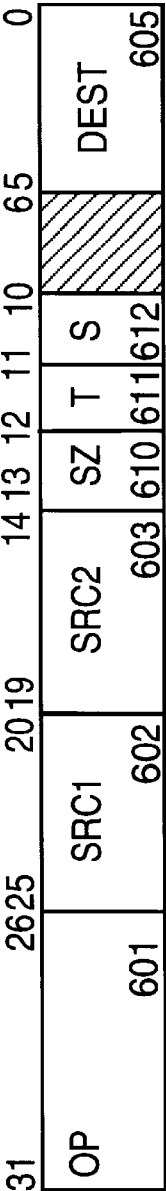


Figure 6a

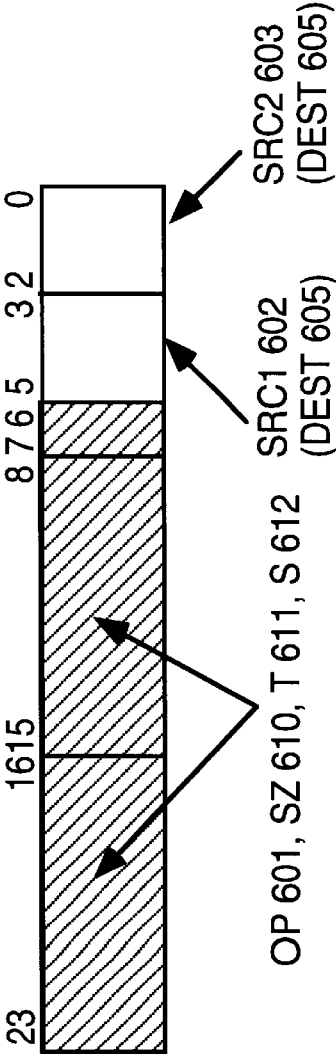


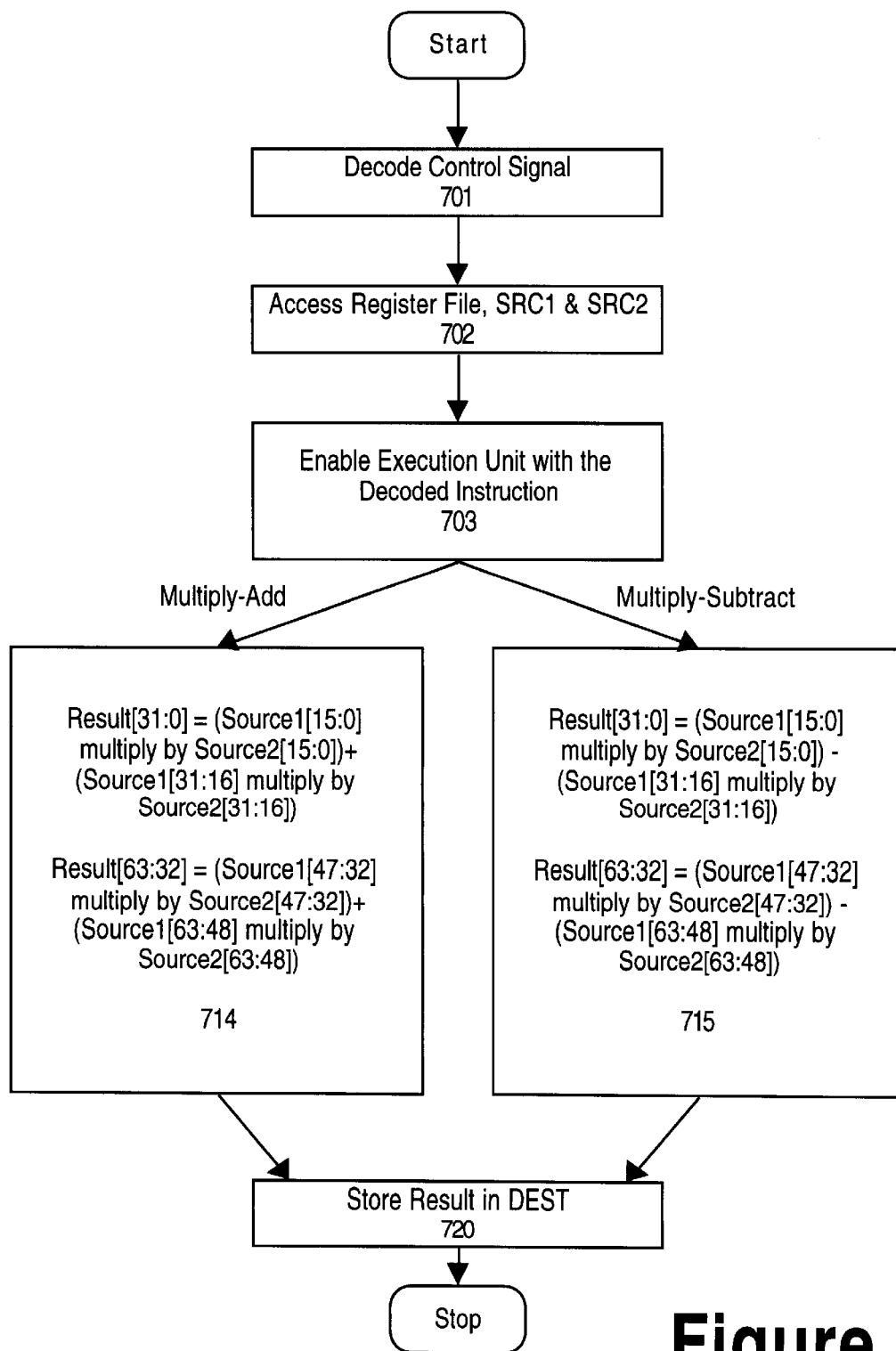
Figure 6b

U.S. Patent

May 7, 2002

Sheet 6 of 7

US 6,385,634 B1

**Figure 7**

U.S. Patent

May 7, 2002

Sheet 7 of 7

US 6,385,634 B1

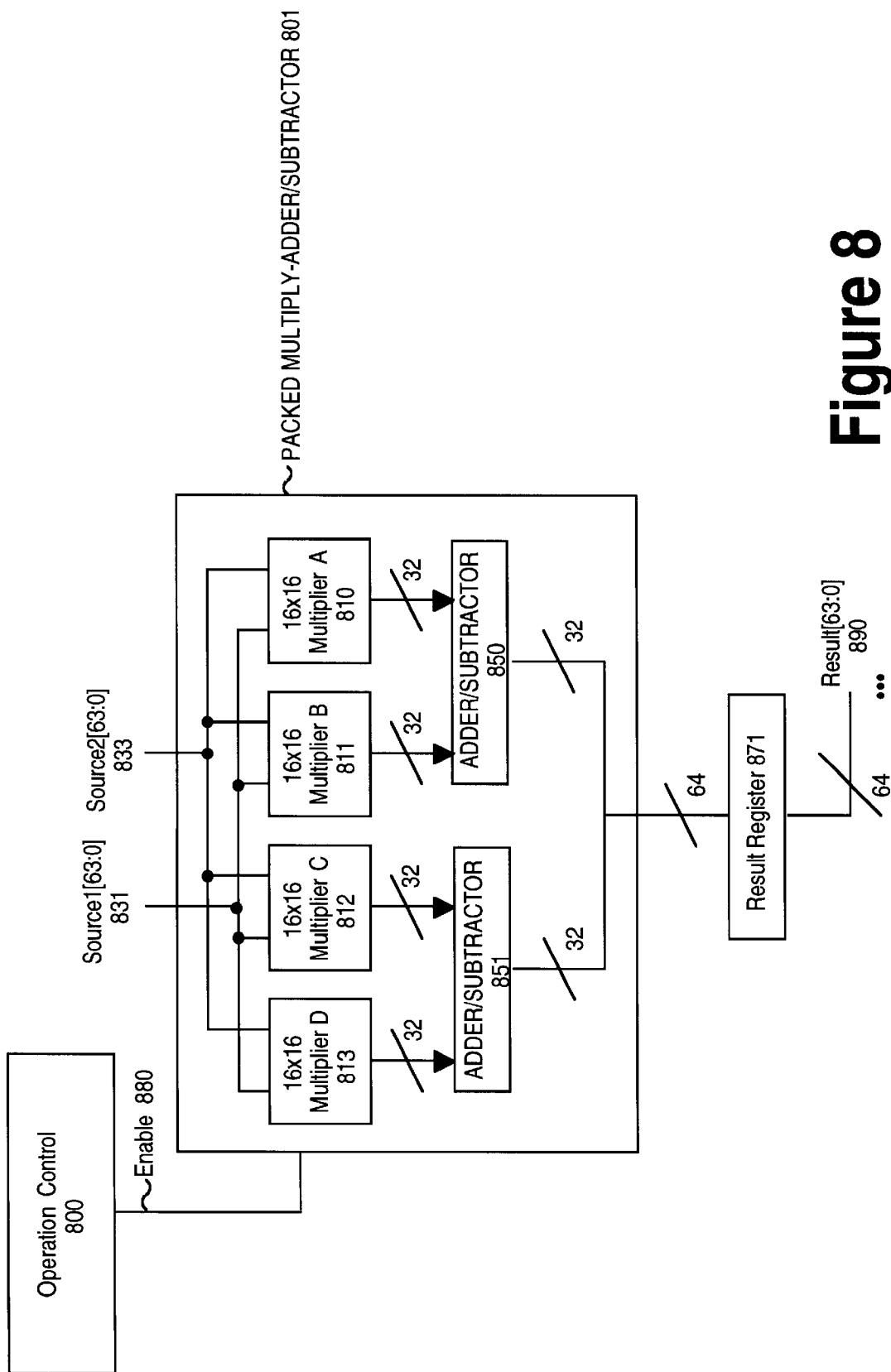


Figure 8

US 6,385,634 B1

1

**METHOD FOR PERFORMING MULTIPLY-
ADD OPERATIONS ON PACKED DATA****CROSS-REFERENCE TO RELATED APPLICATION**

Application Ser. No. 554,625, now U.S. Pat. No. 5,721, 892 titled "Method and Apparatus for Performing Multiply-Subtract Operations on Packed Data," filed Nov. 6, 1995, by Alexander D. Peleg, Millind Mittal, Larry M. Mennemeier, Benny Eitan, Carole Dulong, Eiichi Kowashi, and Wolf Witt which is a continuation of application Ser. No. 521,803, now abandoned, titled "A Method and Apparatus for Performing Multiply-Subtract Operations on Packed Data," filed Aug. 31, 1995.

BACKGROUND OF THE INVENTION**1. Field of Invention**

In particular, the invention relates to the field of computer systems. More specifically, the invention relates to the area of packed data operations.

2. Description of Related Art

In typical computer systems, processors are implemented to operate on values represented by a large number of bits (e.g., 64) using instructions that produce one result. For example, the execution of an add instruction will add together a first 64-bit value and a second 64-bit value and store the result as a third 64-bit value. However, multimedia applications (e.g., applications targeted at computer supported cooperation (CSC—the integration of teleconferencing with mixed media data manipulation), 2D/3D graphics, image processing, video compression/decompression, recognition algorithms and audio manipulation) require the manipulation of large amounts of data which may be represented in a small number of bits. For example, graphical data typically requires 8 or 16 bits and sound data typically requires 8 or 16 bits. Each of these multimedia application requires one or more algorithms, each requiring a number of operations. For example, an algorithm may require an add, compare and shift operation.

To improve efficiency of multimedia applications (as well as other applications that have the same characteristics), prior art processors provide packed data formats. A packed data format is one in which the bits typically used to represent a single value are broken into a number of fixed sized data elements, each of which represents a separate value. For example, a 64-bit register may be broken into two 32-bit elements, each of which represents a separate 32-bit value. In addition, these prior art processors provide instructions for separately manipulating each element in these packed data types in parallel. For example, a packed add instruction adds together corresponding data elements from a first packed data and a second packed data. Thus, if a multimedia algorithm requires a loop containing five operations that must be performed on a large number of data elements, it is desirable to pack the data and perform these operations in parallel using packed data instructions. In this manner, these processors can more efficiently process multimedia applications.

However, if the loop of operations contains an operation that cannot be performed by the processor on packed data (i.e., the processor lacks the appropriate instruction), the data will have to be unpacked to perform the operation. For example, if the multimedia algorithm requires an add operation and the previously described packed add instruction is not available, the programmer must unpack both the first packed data and the second packed data (i.e., separate the elements comprising both the first packed data and the

2

second packed data), add the separated elements together individually, and then pack the results into a packed result for further packed processing. The processing time required to perform such packing and unpacking often negates the performance advantage for which packed data formats are provided. Therefore, it is desirable to incorporate in a computer system a set of packed data instructions that provide all the required operations for typical multimedia algorithms. However, due to the limited die area on today's general purpose microprocessors, the number of instructions which may be added is limited. Therefore, it is desirable to invent instructions that provide both versatility (i.e. instructions which may be used in a wide variety of multimedia algorithms) and the greatest performance advantage.

One prior art technique for providing operations for use in multimedia algorithms is to couple a separate digital signaling processor (DSP) to an existing general purpose processor (e.g., The Intel® 486 manufactured by Intel Corporation of Santa Clara, Calif.). The general purpose processor allocates jobs that can be performed using packed data (e.g., video processing) to the DSP.

One such prior art DSP includes a multiply accumulate instruction that adds to an accumulation value the results of multiplying together two values. (see Kawakami, Yuichi, et al., "A Single-Chip Digital Signal Processor for Voiceband Applications", IEEE International Solid-State Circuits Conference, 1980, pp. 40–41). An example of the multiply accumulate operation for this DSP is shown below in Table 1, where the instruction is performed on the data values A_1 and B_1 accessed as Source1 and Source2, respectively.

TABLE 1

| Multiply-Accumulate Source1, Source2 | | |
|---------------------------------------|---------|--|
| A_1 | Source1 | |
| B_1 | Source2 | |
| = | | |
| $A_1 B_1 + \text{Accumulation Value}$ | Result1 | |

One limitation of this prior art instruction is its limited efficiency—i.e., it only operates on 2 values and an accumulation value. For example, to multiply and accumulate two sets of 2 values requires the following 2 instructions performed serially: 1) multiply accumulate the first value from the first set, the first value from the second set, and an accumulation value of zero to generate an intermediate accumulation value; 2) multiply accumulate the second value from the first set, the second value from the second set, and the intermediate accumulation value to generate the result.

Another prior art DSP includes a multiply accumulate instruction that operates on two sets of two values and an accumulation value (See "Digital Signal Processor with Parallel Multipliers", U.S. Pat. No. 4,771,379—referred to herein as the "Ando et al." reference). An example of the multiply accumulate instruction for this DSP is shown below in Table 2, where the instruction is performed on the data values A_1 , A_2 , B_1 and B_2 accessed as Source 1–4, respectively.

US 6,385,634 B1

3

TABLE 2

| | |
|---|---|
| Source1 <div style="border: 1px solid black; padding: 2px; display: inline-block;">A₁</div> | Source3 <div style="border: 1px solid black; padding: 2px; display: inline-block;">A₂</div> |
| Multiply Accumulate | |
| Source2 <div style="border: 1px solid black; padding: 2px; display: inline-block;">B₁</div> | Source4 <div style="border: 1px solid black; padding: 2px; display: inline-block;">B₂</div> |
| = Result1 | |
| <div style="border: 1px solid black; padding: 2px; display: inline-block;">A₁•B₁ + A₂•B₂ + Accumulation Value</div> | |

Using this prior art technique, two sets of 2 values are multiplied and then added to an accumulation value in one instruction.

This multiply accumulate instruction has limited versatility because it always adds to the accumulation value. As a result, it is difficult to use the instruction for operations other than multiply accumulate. For example, the multiplication of complex numbers is commonly used in multimedia applications. The multiplication of two complex number (e.g., $r_1 i_1$ and $r_2 i_2$) is performed according to the following equation:

$$\text{Real Component} = r_1 \cdot r_2 - i_1 \cdot i_2.$$

$$\text{Imaginary Component} = r_1 \cdot i_2 + r_2 \cdot i_1.$$

This prior art DSP cannot perform the function of multiplying together two complex numbers using one multiply accumulate instruction.

The limitations of this multiply accumulate instruction can be more clearly seen when the result of such a calculation is needed in a subsequent multiplication operation rather than an accumulation. For example, if the real component were calculated using this prior art DSP, the accumulation value would need to be initialized to zero in order to correctly compute the result. Then the accumulation value would again need to be initialized to zero in order to calculate the imaginary component. To perform another complex multiplication on the resulting complex number and a third complex number (e.g., $r_3 i_3$), the resulting complex number must be rescaled and stored into the acceptable memory format and the accumulation value must again be initialized to zero. Then, the complex multiplication can be performed as described above. In each of these operations the ALU, which is devoted to the accumulation value, is superfluous hardware and extra instructions are needed to re-initialize this accumulation value. These extra instructions would otherwise have been unnecessary.

A further limitation of this prior art technique is that the data must be accessed through expensive multi-ported memory. This is because the multipliers are connected directly with data memories. Therefore the amount of parallelism which can be exploited is limited to a small number by the cost of the interconnection, and the fact that this interconnection is not decoupled from the instruction.

The Ando, et al. reference also describes that an alternative to this expensive interconnection is to introduce a delay for each subsequent pair of data to be multiplied. This solution diminishes any performance advantages to those provided by the solution previously shown in Table 1.

Furthermore, the notion of multi-ported memory or of pipelined accesses to memory entails the use of multiple addresses. This explicit use of one address per datum, clearly demonstrates that the critical notion of packed data is not employed in this prior art.

4

SUMMARY OF THE INVENTION

A method and apparatus for including in a processor instructions for performing multiply-add operations on packed data is described. In one embodiment, a processor is coupled to a memory. The memory has stored therein a first packed data and a second packed data. The processor performs operations on data elements in the first packed data and the second packed data to generate a third packed data in response to receiving an instruction. At least two of the data elements in this third packed data storing the result of performing multiply-add operations on data elements in the first and second packed data.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated by way of example, and not limitation, in the figures. Like references indicate similar elements.

FIG. 1 illustrates an exemplary computer system according to one embodiment of the invention.

FIG. 2 illustrates a register file of the processor according to one embodiment of the invention.

FIG. 3 is a flow diagram illustrating the general steps used by the processor to manipulate data according to one embodiment of the invention.

FIG. 4 illustrates packed data-types according to one embodiment of the invention.

FIG. 5a illustrates in-register packed data representations according to one embodiment of the invention.

FIG. 5b illustrates in-register packed data representations according to one embodiment of the invention.

FIG. 5c illustrates in-register packed data representations according to one embodiment of the invention.

FIG. 6a illustrates a control signal format for indicating the use of packed data according to one embodiment of the invention.

FIG. 6b illustrates a second control signal format for indicating the use of packed data according to one embodiment of the invention.

FIG. 7 is a flow diagram illustrating a method for performing multiply-add and multiply-subtract operations on packed data according to one embodiment of the invention.

FIG. 8 illustrates a circuit for performing multiply-add and/or multiply-subtract operations on packed data according to one embodiment of the invention.

DETAILED DESCRIPTION

In the following description, numerous specific details are set forth to provide a thorough understanding of the invention. However, it is understood that the invention may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in detail in order not to obscure the invention.

Definitions

To provide a foundation for understanding the description of the embodiments of the invention, the following definitions are provided.

Bit X through Bit Y:

defines a subfield of binary number. For example, bit six through bit zero of the byte 00111010_2 (shown in base two) represent the subfield 111010_2 . The '2' following a binary number indicates base 2. Therefore, 1000_2 equals 8_{10} , while F_{16} equals 15_{10} .

US 6,385,634 B1

5

Rx:

is a register. A register is any device capable of storing and providing data. Further functionality of a register is described below. A register is not necessarily, included on the same die or in the same package as the processor.

SRC1, SRC2, and DEST:

identify storage areas (e.g., memory addresses, registers, etc.)

Source1-i and Result1-i:

represent data.

Overview

This application describes a method and apparatus for including in a processor instructions for performing multiply-add and multiply-subtract operations on packed data. In one embodiment, two multiply-add operations are performed using a single multiply-add instruction as shown below in Table 3a and Table 3b—Table 3a shows a simplified representation of the disclosed multiply-add instruction, while Table 3b shows a bit level example of the disclosed multiply-add instruction.

TABLE 3a

| Multiply-Add Source1, Source2 | | | | |
|---|----------------|---|----------------|---------|
| A ₁ | A ₂ | A ₃ | A ₄ | Source1 |
| B ₁ | B ₂ | B ₃ | B ₄ | Source2 |
| A ₁ B ₁ + A ₂ B ₂ | | A ₃ B ₃ + A ₄ B ₄ | | Result1 |

TABLE 3b

| | | | |
|------------------------------|------------------------------|------------------------------|------------------------------|
| 11111111 11111111 | 11111111 00000000 | 01110001 11000111 | 01110001 11000111 |
| 3 | 2 | 1 | 0 |
| Multiply | Multiply | Multiply | Multiply |
| 00000000 00000000 | 00000000 00000001 | 10000000 00000000 | 00000100 00000000 |
| ↓ | ↓ | ↓ | ↓ |
| 32-Bit Intermediate Result 4 | 32-Bit Intermediate Result 3 | 32-Bit Intermediate Result 2 | 32-Bit Intermediate Result 1 |
| Add | | Add | |
| 11111111 11111111 | 11111111 00000000 | 11001000 11100011 | 10011100 00000000 |
| 1 | | 0 | |

Thus, the described embodiment of the multiple-add instruction multiplies together corresponding 16-bit data elements of Source1 and Source2 generating four 32-bit intermediate results. These 32-bit intermediate results are summed by pairs producing two 32-bit results that are packed into their respective elements of a packed result. As further described later, alternative embodiment may vary the number of bits in the data elements, intermediate results, and results. In addition, alternative embodiment may vary the number of data elements used, the number of intermediate results generated, and the number of data elements in the resulting packed data. The multiply-subtract operation is the

6

same as the multiply-add operation, except the adds are replaced with subtracts. The operation of an example multiply-subtract instruction is shown below in Table 4.

TABLE 4

| Multiply-Subtract Source1, Source2 | | | | |
|---|----------------|---|----------------|---------|
| A ₁ | A ₂ | A ₃ | A ₄ | Source1 |
| B ₁ | B ₂ | B ₃ | B ₄ | Source2 |
| A ₁ B ₁ - A ₂ B ₂ | | A ₃ B ₃ - A ₄ B ₄ | | Result1 |

Of course, alternative embodiments may implement variations of these instructions. For example, alternative embodiments may include an instruction which performs at least one multiply-add operation or at least one multiply-subtract operation. As another example, alternative embodiments may include an instruction which performs at least one multiply-add operation in combination with at least one multiply-subtract operation. As another example, alternative embodiments may include an instruction which perform multiply-add operation(s) and/or multiply-subtract operation(s) in combination with some other operation.

Computer System

FIG. 1 illustrates an exemplary computer system 100 according to one embodiment of the invention. Computer system 100 includes a bus 101, or other communications hardware and software, for communicating information, and a processor 109 coupled with bus 101 for processing information. Processor 109 represents a central processing unit of any type of architecture, including a CISC or RISC type

architecture. Computer system 100 further includes a random access memory (RAM) or other dynamic storage device (referred to as main memory 104), coupled to bus 101 for storing information and instructions to be executed by processor 109. Main memory 104 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 109. Computer system 100 also includes a read only memory (ROM) 106, and/or other static storage device, coupled to bus 101 for storing static information and instructions for processor 109. Data storage device 107 is coupled to bus 101 for storing information and instructions.

US 6,385,634 B1

7

FIG. 1 also illustrates that processor 109 includes an execution unit 130, a register file 150, a cache 160, a decoder 165, and an internal bus 170. Of course, processor 109 contains additional circuitry which is not necessary to understanding the invention.

Execution unit 130 is used for executing instructions received by processor 109. In addition to recognizing instructions typically implemented in general purpose processors, execution unit 130 recognizes instructions in packed instruction set 140 for performing operations on packed data formats. Packed instruction set 140 includes instructions for supporting multiply-add and/or multiply-subtract operations. In addition, packed instruction set 140 may also include instructions for supporting a pack operation, an unpack operation, a packed add operation, a packed subtract operation, a packed multiply operation, a packed shift operation, a packed compare operation, a population count operation, and a set of packed logical operations (including packed AND, packed ANDNOT, packed OR, and packed XOR) as described in "A Set of Instructions for Operating on Packed Data," filed on Aug. 31, 1995, Ser. No. 08/521,360 now abandoned.

Execution unit 130 is coupled to register file 150 by internal bus 170. Register file 150 represents a storage area on processor 109 for storing information, including data. It is understood that one aspect of the invention is the described instruction set for operating on packed data. According to this aspect of the invention, the storage area used for storing the packed data is not critical. However, one embodiment of the register file 150 is later described with reference to FIG. 2. Execution unit 130 is coupled to cache 160 and decoder 165. Cache 160 is used to cache data and/or control signals from, for example, main memory 104. Decoder 165 is used for decoding instructions received by processor 109 into control signals and/or microcode entry points. In response to these control signals and/or microcode entry points, execution unit 130 performs the appropriate operations. For example, if an add instruction is received, decoder 165 causes execution unit 130 to perform the required addition; if a subtract instruction is received, decoder 165 causes execution unit 130 to perform the required subtraction; etc. Decoder 165 may be implemented using any number of different mechanisms (e.g., a look-up table, a hardware implementation, a PLA, etc.). Thus, while the execution of the various instructions by the decoder and execution unit is represented by a series of if/then statements, it is understood that the execution of an instruction does not require a serial processing of these if/then statements. Rather, any mechanism for logically performing this if/then processing is considered to be within the scope of the invention.

FIG. 1 additionally shows a data storage device 107, such as a magnetic disk or optical disk, and its corresponding disk drive, can be coupled to computer system 100. Computer system 100 can also be coupled via bus 101 to a display device 121 for displaying information to a computer user. Display device 121 can include a frame buffer, specialized graphics rendering devices, a cathode ray tube (CRT), and/or a flat panel display. An alphanumeric input device 122, including alphanumeric and other keys, is typically coupled to bus 101 for communicating information and command selections to processor 109. Another type of user input device is cursor control 123, such as a mouse, a trackball, a pen, a touch screen, or cursor direction keys for communicating direction information and command selections to processor 109, and for controlling cursor movement on display device 121. This input device typically has two

8

degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), which allows the device to specify positions in a plane. However, this invention should not be limited to input devices with only two degrees of freedom.

Another device which may be coupled to bus 101 is a hard copy device 124 which may be used for printing instructions, data, or other information on a medium such as paper, film, or similar types of media. Additionally, computer system 100 can be coupled to a device for sound recording, and/or playback 125, such as an audio digitizer coupled to a microphone for recording information. Further, the device may include a speaker which is coupled to a digital to analog (D/A) converter for playing back the digitized sounds.

Also, computer system 100 can be a terminal in a computer network (e.g., a LAN). Computer system 100 would then be a computer subsystem of a computer network. Computer system 100 optionally includes video digitizing device 126. Video digitizing device 126 can be used to capture video images that can be transmitted to others on the computer network.

In one embodiment, the processor 109 additionally supports an instruction set which is compatible with the x86 instruction set used by existing processors (such as the Pentium® processor) manufactured by Intel Corporation of Santa Clara, Calif. Thus, in one embodiment, processor 109 supports all the operations supported in the IA™—Intel Architecture, as defined by Intel Corporation of Santa Clara, Calif. (see Microprocessors, Intel Data Books volume 1 and volume 2, 1992 and 1993, available from Intel of Santa Clara, Calif.). As a result, processor 109 can support existing x86 operations in addition to the operations of the invention. While the invention is described as being incorporated into an x86 based instruction set, alternative embodiments could incorporate the invention into other instruction sets. For example, the invention could be incorporated into a 64-bit processor using a new instruction set.

FIG. 2 illustrates the register file of the processor according to one embodiment of the invention. The register file 150 is used for storing information, including control/status information, integer data, floating point data, and packed data. In the embodiment shown in FIG. 2, the register file 150 includes integer registers 201, registers 209, status registers 208, and instruction pointer register 211. Status registers 208 indicate the status of processor 109. Instruction pointer register 211 stores the address of the next instruction to be executed. Integer registers 201, registers 209, status registers 208, and instruction pointer register 211 are all coupled to internal bus 170. Any additional registers would also be coupled to internal bus 170.

In one embodiment, the registers 209 are used for both packed data and floating point data. In one such embodiment, the processor 109, at any given time, must treat the registers 209 as being either stack referenced floating point registers or non-stack referenced packed data registers. In this embodiment, a mechanism is included to allow the processor 109 to switch between operating on registers 209 as stack referenced floating point registers and non-stack referenced packed data registers. In another such embodiment, the processor 109 may simultaneously operate on registers 209 as non-stack referenced floating point and packed data registers. As another example, in another embodiment, these same registers may be used for storing integer data.

Of course, alternative embodiments may be implemented to contain more or less sets of registers. For example, an

US 6,385,634 B1

9

alternative embodiment may include a separate set of floating point registers for storing floating point data. As another example, an alternative embodiment may including a first set of registers, each for storing control/status information, and a second set of registers, each capable of storing integer, floating point, and packed data. As a matter of clarity, the registers of an embodiment should not be limited in meaning to a particular type of circuit. Rather, a register of an embodiment need only be capable of storing and providing data, and performing the functions described herein.

The various sets of registers (e.g., the integer registers **201**, the registers **209**) may be implemented to include different numbers of registers and/or to different size registers. For example, in one embodiment, the integer registers **201** are implemented to store thirty-two bits, while the registers **209** are implemented to store eighty bits (all eighty bits are used for storing floating point data, while only sixty-four are used for packed data). In addition, registers **209** contains eight registers, R_0 **212a** through R_7 **212h**. R_1 **212a**, R_2 **212b** and R_3 **212c** are examples of individual registers in registers **209**. Thirty-two bits of a register in registers **209** can be moved into an integer register in integer registers **201**. Similarly, a value in an integer register can be moved into thirty-two bits of a register in registers **209**. In another embodiment, the integer registers **201** each contain 64 bits, and 64 bits of data may be moved between the integer register **201** and the registers **209**.

FIG. 3 is a flow diagram illustrating the general steps are used by the processor to manipulate data according to one embodiment of the invention. That is, FIG. 3 illustrates the steps followed by processor **109** while performing an operation on packed data, performing an operation on unpacked data, or performing some other operation. For example, such operations include a load operation to load a register in register file **150** with data from cache **160**, main memory **104**, read only memory (ROM) **106**, or data storage device **107**.

At step **301**, the decoder **165** receives a control signal from either the cache **160** or bus **101**. Decoder **165** decodes the control signal to determine the operations to be performed.

At step **302**, Decoder **165** accesses the register file **150**, or a location in memory. Registers in the register file **150**, or memory locations in the memory, are accessed depending on the register address specified in the control signal. For example, for an operation on packed data, the control signal can include SRC1, SRC2 and DEST register addresses. SRC1 is the address of the first source register. SRC2 is the address of the second source register. In some cases, the SRC2 address is optional as not all operations require two source addresses. If the SRC2 address is not required for an operation, then only the SRC1 address is used. DEST is the address of the destination register where the result data is stored. In one embodiment, SRC1 or SRC2 is also used as DEST. SRC1, SRC2 and DEST are described more fully in relation to FIG. 6a and FIG. 6b. The data stored in the corresponding registers is referred to as Source1, Source2, and Result respectively. Each of these data is sixty-four bits in length.

In another embodiment of the invention, any one, or all, of SRC1, SRC2 and DEST, can define a memory location in the addressable memory space of processor **109**. For example, SRC1 may identify a memory location in main memory **104**, while SRC2 identifies a first register in integer registers **201** and DEST identifies a second register in registers **209**. For simplicity of the description herein, the

10

invention will be described in relation to accessing the register file **150**. However, these accesses could be made to memory instead.

At step **303**, execution unit **130** is enabled to perform the operation on the accessed data. At step **304**, the result is stored back into register file **150** according to requirements of the control signal.

Data and Storage Formats

FIG. 4 illustrates packed data-types according to one embodiment of the invention. Three packed data formats are illustrated; packed byte **401**, packed word **402**, and packed doubleword **403**. Packed byte, in one embodiment of the invention, is sixty-four bits long containing eight data elements. Each data element is one byte long. Generally, a data element is an individual piece of data that is stored in a single register (or memory location) with other data elements of the same length. In one embodiment of the invention, the number of data elements stored in a register is sixty-four bits divided by the length in bits of a data element.

Packed word **402** is sixty-four bits long and contains four word **402** data elements. Each word **402** data element contains sixteen bits of information.

Packed doubleword **403** is sixty-four bits long and contains two doubleword **403** data elements. Each doubleword **403** data element contains thirty-two bits of information.

FIGS. 5a through 5c illustrate the in-register packed data storage representation according to one embodiment of the invention. Unsigned packed byte in-register representation **510** illustrates the storage of an unsigned packed byte **401** in one of the registers R_0 **212a** through R_7 **212h**. Information for each byte data element is stored in bit seven through bit zero for byte zero, bit fifteen through bit eight for byte one, bit twenty-three through bit sixteen for byte two, bit thirty-one through bit twenty-four for byte three, bit thirty-nine through bit thirty-two for byte four, bit forty-seven through bit forty for byte five, bit fifty-five through bit forty-eight for byte six and bit sixty-three through bit fifty-six for byte seven. Thus, all available bits are used in the register. This storage arrangement increases the storage efficiency of the processor. As well, with eight data elements accessed, one operation can now be performed on eight data elements simultaneously. Signed packed byte in-register representation **511** illustrates the storage of a signed packed byte **401**. Note that the eighth bit of every byte data element is the sign indicator.

Unsigned packed word in-register representation **512** illustrates how word three through word zero are stored in one register of registers **209**. Bit fifteen through bit zero contain the data element information for word zero, bit thirty-one through bit sixteen contain the information for data element word one, bit forty-seven through bit thirty-two contain the information for data element word two and bit sixty-three through bit forty-eight contain the information for data element word three. Signed packed word in-register representation **513** is similar to the unsigned packed word in-register representation **512**. Note that the sixteenth bit of each word data element is the sign indicator.

Unsigned packed doubleword in-register representation **514** shows how registers **209** store two doubleword data elements. Doubleword zero is stored in bit thirty-one through bit zero of the register. Doubleword one is stored in bit sixty-three through bit thirty-two of the register. Signed packed doubleword in-register representation **515** is similar to unsigned packed doubleword in-register representation

US 6,385,634 B1

11

514. Note that the necessary sign bit is the thirty-second bit of the doubleword data element.

As mentioned previously, registers **209** may be used for both packed data and floating point data. In this embodiment of the invention, the individual programming processor **109** may be required to track whether an addressed register, R_0 **212a** for example, is storing packed data or floating point data. In an alternative embodiment, processor **109** could track the type of data stored in individual registers of registers **209**. This alternative embodiment could then generate errors if, for example, a packed addition operation were attempted on floating point data.

Control Signal Formats

The following describes one embodiment of the control signal formats used by processor **109** to manipulate packed data. In one embodiment of the invention, control signals are represented as thirty-two bits. Decoder **165** may receive the control signal from bus **101**. In another embodiment, decoder **165** can also receive such control signals from cache **160**.

FIG. **6a** illustrates a control signal format for indicating the use of packed data according to one embodiment of the invention. Operation field **OP 601**, bit thirty-one through bit twenty-six, provides information about the operation to be performed by processor **109**; for example, packed addition, packed subtraction, etc. **SRC1 602**, bit twenty-five through twenty, provides the source register address of a register in registers **209**. This source register contains the first packed data, **Source1**, to be used in the execution of the control signal. Similarly, **SRC2 603**, bit nineteen through bit fourteen, contains the address of a register in registers **209**. This second source register contains the packed data, **Source2**, to be used during execution of the operation. **DEST 605**, bit five through bit zero, contains the address of a register in registers **209**. This destination register will store the result packed data, **Result**, of the packed data operation.

Control bits **SZ 610**, bit twelve and bit thirteen, indicates the length of the data elements in the first and second packed data source registers. If **SZ 610** equals 01_2 , then the packed data is formatted as packed byte **401**. If **SZ 610** equals 10_2 , then the packed data is formatted as packed word **402**. **SZ 610** equaling 00_2 or 11_2 is reserved, however, in another embodiment, one of these values could be used to indicate packed doubleword **403**.

Control bit **T 611**, bit eleven, indicates whether the operation is to be carried out with saturate mode. If **T 611** equals one, then a saturating operation is performed. If **T 611** equals zero, then a non-saturating operation is performed. Saturating operations will be described later.

Control bit **S 612**, bit ten, indicates the use of a signed operation. If **S 612** equals one, then a signed operation is performed. If **S 612** equals zero, then an unsigned operation is performed.

FIG. **6b** illustrates a second control signal format for indicating the use of packed data according to one embodiment of the invention. This format corresponds with the general integer opcode format described in the "Pentium Processor Family User's Manual," available from Intel Corporation, Literature Sales, P.O. Box 7641, Mt. Prospect, Ill., 60056-7641. Note that **OP 601**, **SZ 610**, **T 611**, and **S 612** are all combined into one large field. For some control signals, bits three through five are **SRC1 602**. In one embodiment, where there is a **SRC1 602** address, then bits three through five also correspond to **DEST 605**. In an alternate embodiment, where there is a **SRC2 603** address,

12

then bits zero through two also correspond to **DEST 605**. For other control signals, like a packed shift immediate operation, bits three through five represent an extension to the opcode field. In one embodiment, this extension allows a programmer to include an immediate value with the control signal, such as a shift count value. In one embodiment, the immediate value follows the control signal. This is described in more detail in the "Pentium Processor Family User's Manual," in appendix F, pages F-1 through F-3. Bits zero through two represent **SRC2 603**. This general format allows register to register, memory to register, register by memory, register by register, register by immediate, register to memory addressing. Also, in one embodiment, this general format can support integer register to register, and register to integer register addressing.

Description of Saturate/Unsaturate

As mentioned previously, **T 611** indicates whether operations optionally saturate. Where the result of an operation, with saturate enabled, overflows or underflows the range of the data, the result will be clamped. Clamping means setting the result to a maximum or minimum value should a result exceed the range's maximum or minimum value. In the case of underflow, saturation clamps the result to the lowest value in the range and in the case of overflow, to the highest value. The allowable range for each data format is shown in Table 5.

TABLE 5

| Data Format | Minimum Value | Maximum Value |
|---------------------|---------------|---------------|
| Unsigned Byte | 0 | 255 |
| Signed Byte | -128 | 127 |
| Unsigned Word | 0 | 65535 |
| Signed Word | -32768 | 32767 |
| Unsigned Doubleword | 0 | $2^{64} - 1$ |
| Signed Doubleword | -2^{63} | $2^{63} - 1$ |

As mentioned above, **T 611** indicates whether saturating operations are being performed. Therefore, using the unsigned byte data format, if an operation's result=258 and saturation was enabled, then the result would be clamped to 255 before being stored into the operation's destination register. Similarly, if an operation's result=-32999 and processor **109** used signed word data format with saturation enabled, then the result would be clamped to -32768 before being stored into the operation's destination register.

Multiply-Add/Subtract Operation(S)

In one embodiment of the invention, the **SRC1** register contains packed data (**Source1**), the **SRC2** register contains packed data (**Source2**), and the **DEST** register will contain the result (**Result**) of performing the multiply-add or multiply-subtract instruction on **Source1** and **Source2**. In the first step of the multiply-add and multiply-subtract instruction, **Source1** will have each data element independently multiplied by the respective data element of **Source2** to generate a set of respective intermediate results. These intermediate results are summed by pairs to generate the **Result** for the multiply-add instruction. In contrast, these intermediate results are subtracted by pairs to generate the **Result** for the multiply-subtract instruction.

In one embodiment of the invention, the multiply-add and multiply-subtract instructions operate on signed packed data and truncate the results to avoid any overflows. In addition, these instructions operate on packed word data and the

US 6,385,634 B1

13

Result is a packed double word. However, alternative embodiments could support these instructions for other packed data types.

FIG. 7 is a flow diagram illustrating a method for performing multiply-add and multiply-subtract operations on packed data according to one embodiment of the invention.

At step 701, decoder 165 decodes the control signal received by processor 109. Thus, decoder 165 decodes: the operation code for a multiply-add instruction or a multiply-subtract instruction.

At step 702, via internal bus 170, decoder 165 accesses registers 209 in register file 150 given the SRC1 602 and SRC2 603 addresses. Registers 209 provide execution unit 130 with the packed data stored in the SRC1 602 register (Source1), and the packed data stored in SRC2 603 register (Source2). That is, registers 209 communicate the packed data to execution unit 130 via internal bus 170.

At step 703, decoder 165 enables execution unit 130 to perform the instruction. If the instruction is a multiply-add instruction, flow passes to step 714. However, if the instruction is a multiply-subtract instruction, flow passes to step 715.

In step 714, the following is performed. Source1 bits fifteen through zero are multiplied by Source2 bits fifteen through zero generating a first 32-bit intermediate result (Intermediate Result 1). Source1 bits thirty-one through sixteen are multiplied by Source2 bits thirty-one through sixteen generating a second 32-bit intermediate result (Intermediate Result 2). Source1 bits forty-seven through thirty-two are multiplied by Source2 bits forty-seven through thirty-two generating a third 32-bit intermediate result (Intermediate Result 3). Source1 bits sixty-three through forty-eight are multiplied by Source2 bits sixty-three through forty-eight generating a fourth 32-bit intermediate result (Intermediate Result 4). Intermediate Result 1 is added to Intermediate Result 2 generating Result bits thirty-one through 0, and Intermediate Result 3 is added to Intermediate Result 4 generating Result bits sixty-three through thirty-two. Step 715 is the same as step 714, with the exception that Intermediate Result 1 Intermediate Result 2 are subtracted to generate bits thirty-one through 0 of the Result, and Intermediate Result 3 and Intermediate Result 4 are subtracted to generate bits sixty-three through thirty-two of the Result.

Different embodiments may perform the multiplies and adds/subtracts serially, in parallel, or in some combination of serial and parallel operations.

At step 720, the Result is stored in the DEST register.

Packed Data Multiply-Add/Subtract Circuits

In one embodiment, the multiply-add and multiply-subtract instructions can execute on multiple data elements in the same number of clock cycles as a single multiply on unpacked data. To achieve execution in the same number of clock cycles, parallelism is used. That is, registers are simultaneously instructed to perform the multiply-add/subtract operations on the data elements. This is discussed in more detail below.

FIG. 8 illustrates a circuit for performing multiply-add and/or multiply-subtract operations on packed data according to one embodiment of the invention. Operation control 800 processes the control signal for the multiply-add and multiply-subtract instructions. Operation control 800 outputs signals on Enable 880 to control Packed multiply-adder/subtractor 801.

14

Packed multiply-adder/subtractor 801 has the following inputs: Source1[63:0] 831, Source2[63:0] 833, and Enable 880. Packed multiply-adder/subtractor 801 includes four 16×16 multiplier circuits: 16×16 multiplier A 810, 16×16 multiplier B 811, 16×16 multiplier C 812 and 16×16 multiplier D 813. 16×16 multiplier A 810 has as inputs Source1 [15:0] and Source2[15:0]. 16×16 multiplier B 811 has as inputs Source1[31:16] and Source2[31:16]. 16×16 multiplier C 812 has as inputs Source1[47:32] and Source2 [47:32]. 16×16 multiplier D 813 has as inputs Source1 [63:48] and Source2[63:48]. The 32-bit intermediate results generated by 16×16 multiplier A 810 and 16×16 multiplier B 811 are received by adder/subtractor 1350, while the 32-bit intermediate results generated by 16×16 multiplier C 812 and 16×16 multiplier D 813 are received by adder/subtractor 851.

Based on whether the current instruction is a multiply/add or multiply/subtract instruction, adder/subtractor 850 and adder/subtractor 851 add or subtract their respective 32-bit inputs. The output of adder/subtractor 850 (i.e., Result bits 31 through zero of the Result) and the output of adder/subtractor 851 (i.e., bits 63 through 32 of the Result) are combined into the 64-bit Result and communicated to Result Register 871.

In one embodiment, each of adder/subtractor 851 and adder/subtractor 850 are composed of four 8-bit adders/subtractors with the appropriate propagation delays. However, alternative embodiments could implement adder/subtractor 851 and adder/subtractor 850 in any number of ways (e.g., two 32-bit adders/subtractors).

To perform the equivalent of these multiply-add or multiply-subtract instructions in prior art processors which operate on unpacked data, four separate 64-bit multiply operations and two 64-bit add or subtract operations, as well as the necessary load and store operations, would be needed. This wastes data lines and circuitry that are used for the bits that are higher than bit sixteen for Source1 and Source2, and higher than bit thirty two for the Result. As well, the entire 64-bit result generated by the prior art processor may not be of use to the programmer. Therefore, the programmer would have to truncate each result.

Performing the equivalent of this multiply-add instruction using the prior art DSP processor described with reference to Table 1 requires one instruction to zero the accumulation value and four multiply accumulate instructions. Performing the equivalent of this multiply-add instruction using the prior art DSP processor described with reference to Table 2 requires one instruction to zero the accumulation value and 2-accumulate instructions.

Advantages of Including the Described Multiply-Add Instruction in the Instruction Set

As previously described, the prior art multiply accumulate instructions always add the results of their multiplications to an accumulation value. This accumulation value becomes a bottleneck for performing operations other than multiplying and accumulating (e.g., the accumulation value must be cleared each time a new set of operations is required which do not require the previous accumulation value). This accumulation value also becomes a bottleneck if operations, such as rounding, need to be performed before accumulation.

In contrast, the disclosed multiply-add and multiply-subtract instructions do not carry forward an accumulation value. As a result, these instructions are easier to use in a wider variety of algorithms. In addition, software pipelining can be used to achieve comparable throughput. To illustrate

15

the versatility of the multiply-add instruction, several example multimedia algorithms are described below. Some of these multimedia algorithms use additional packed data instructions. The operation of these additional packed data instructions are shown in relation to the described algorithms. For a further description of these packed data instructions, see "A Set of Instructions for Operating on Packed Data," filed on Aug. 31, 1995, Ser. No. 08/521,360 now abandoned. Of course, other packed data instructions could be used. In addition, a number of steps requiring the use of general purpose processor instructions to manage data movement, looping, and conditional branching have been omitted in the following examples.

1) Multiplication of Complex Numbers

The disclosed multiply-add instruction can be used to multiply two complex numbers in a single instruction as shown in Table 6a. As previously described, the multiplication of two complex number (e.g., $r_1 i_1$ and $r_2 i_2$) is performed according to the following equation:

$$\text{Real Component} = r_1 \cdot r_2 - i_1 \cdot i_2.$$

Imaginary Component= $r_1 \cdot i_2 + r_2 \cdot i_1$

If this instruction is implemented to be completed every clock cycle, the invention can multiply two complex numbers every clock cycle.

TABLE 6a

| Multiply-Add Source1, Source2 | | | | |
|---|----------------|----------------|---|---------|
| r ₁ | i ₂ | r ₁ | i ₁ | Source1 |
| r ₂ | i ₁ | i ₂ | r ₂ | Source2 |
| Real Component: | | = | Imaginary Component: Result | |
| r ₁ r ₂ - i ₁ i ₂ | | | r ₁ i ₁ + r ₂ i ₁ | |

As another example, Table 6b shows the instructions used to multiply together three complex numbers.

TABLE 6b

| Multiply-Add Source1, Source2 | | | | |
|-------------------------------------|------------------------------------|------------------------------------|------------------------------------|---------------------|
| r_1 | i_2 | r_1 | i_1 | Source1 |
| r_2 | $-i_1$ | i_2 | r_2 | Source2 |
| Real Component ₁ : | | Imaginary Component ₁ : | | Result |
| $r_1 r_2 - i_1 i_2$ | | $r_1 i_2 + r_2 i_1$ | | |
| Packed Shift Right Source1, Source2 | | | | |
| Real Component ₁ | | Imaginary Component ₁ | | Result1 |
| | | 16 | | |
| | | = | | |
| Real Component ₁ | | Imaginary Component ₁ | | Result2 |
| Pack Result2, Result2 | | | | |
| Real Component ₁ | | Imaginary Component ₁ | | Result2 |
| Real Component ₁ | | Imaginary Component ₁ | | Result2 |
| | | = | | |
| Real Component ₁ | Imaginary Component ₁ | Real Component ₁ | Imaginary Component ₁ | Result ₃ |
| Multiply-Add Result3, Source3 | | | | |
| Real Component ₁ : | Imaginary Component ₁ : | Real Component ₁ : | Imaginary Component ₁ : | Result3 |

16

TABLE 6b-continued

| | | | | | |
|---------------------------------|----------------------------------|-----|----------------------------------|---------------------------------|---------------------|
| $\frac{r_1 r_2 - i_1 i_2}{r_3}$ | $\frac{r_1 i_2 + r_2 i_1}{-i_3}$ | $=$ | $\frac{r_1 r_2 - i_1 i_2}{i_3}$ | $\frac{r_1 i_2 + r_2 i_1}{r_3}$ | Source3 |
| Real Component ₂ | | | Imaginary Component ₂ | | Result ₄ |

2) Multiply Accumulation Operations

The disclosed multiply-add instructions can also be used to multiply and accumulate values. For example, two sets of four data elements (A_{1-4} and B_{1-4}) may be multiplied and accumulated as shown below in Table 7. In one embodiment, each of the instructions shown in Table 7 is implemented to complete each clock cycle.

TABLE 7

| Multiply-Add Source1, Source2 | | | | |
|-------------------------------|---|-------------------------------------|-------|---------|
| 0 | 0 | A_1 | A_2 | Source1 |
| 0 | 0 | B_1 | B_2 | Source2 |
| | | = | | |
| 0 | | $A_1B_1 + A_2B_2$ | | Result1 |
| Multiply-Add Source3, Source4 | | | | |
| 0 | 0 | A_3 | A_4 | Source3 |
| 0 | 0 | B_3 | B_4 | Source4 |
| | | = | | |
| 0 | | $A_3A_4 + B_3B_4$ | | Result2 |
| Unpacked Add Result1, Result2 | | | | |
| 0 | | $A_1B_1 + A_2B_2$ | | Result1 |
| 0 | | $A_3A_4 + B_3B_4$ | | Result2 |
| | | = | | |
| 0 | | $A_1B_1 + A_2B_2 + A_3A_4 + B_3B_4$ | | Result3 |

If the number of data elements in each set exceeds 8 and is a multiple of 4, the multiplication and accumulation of these sets requires fewer instructions if performed as shown in table 8 below.

TABLE 8

| | | | | |
|-------------------------------------|-------------------------------------|-------------------------------------|-------|---------|
| Multiply-Add Source1, Source2 | | | | |
| A_1 | A_2 | A_3 | A_4 | Source1 |
| B_1 | B_2 | B_3 | B_4 | Source2 |
| | | = | | |
| $A_1B_1 + A_2B_2$ | | $A_3B_3 + A_4B_4$ | | Result1 |
| Multiply-Add Source3, Source4 | | | | |
| A_5 | A_6 | A_7 | A_8 | Source3 |
| B_5 | B_6 | B_7 | B_8 | Source4 |
| | | = | | |
| $A_5B_5 + A_6B_6$ | | $A_7B_7 + A_8B_8$ | | Result2 |
| Packed Add Result1, Result2 | | | | |
| $A_1B_1 + A_2B_2$ | | $A_3B_3 + A_4B_4$ | | Result1 |
| $A_5B_5 + A_6B_6$ | | $A_7B_7 + A_8B_8$ | | Result2 |
| | | = | | |
| $A_1B_1 + A_2B_2 + A_5B_5 + A_6B_6$ | | $A_3B_3 + A_4B_4 + A_7B_7 + A_8B_8$ | | Result3 |
| Unpack High Result3, Source5 | | | | |
| $A_1B_1 + A_2B_2 + A_5B_5 + A_6B_6$ | | $A_3B_3 + A_4B_4 + A_7B_7 + A_8B_8$ | | Result3 |
| 0 | | 0 | | Source5 |
| | | = | | |
| 0 | $A_1B_1 + A_2B_2 + A_5B_5 + A_6B_6$ | | | |
| Unpack Low Result3, Source5 | | | | |

US 6,385,634 B1

17

TABLE 8-continued

| | | |
|-------------------------------------|-------------------------------------|--------------------|
| $A_1B_1 + A_2B_2 + A_5B_5 + A_6B_6$ | $A_3B_3 + A_4B_4 + A_7B_7 + A_8B_8$ | Result3 Source5 |
| 0 | 0 | |
| = | | |
| 0 | $A_3B_3 + A_4B_4 + A_7B_7 + A_8B_8$ | Result5 |
| Packed Add Result4, Result5 | | |
| 0 | $A_1B_1 + A_2B_2 + A_5B_5 + A_6B_6$ | Result4 |
| 0 | $A_3B_3 + A_4B_4 + A_7B_7 + A_8B_8$ | Result5 |
| = | | |
| 0 | TOTAL | Result6 |

As another example, Table 9 shows the separate multiplication and accumulation of sets A and B and sets C and D, where each of these sets includes 2 data elements.

TABLE 9

| Multiply-Add Source1, Source2 | | | | |
|-------------------------------|-------|-------------------|-------|---------|
| A_1 | A_2 | C_1 | C_2 | Source1 |
| B_1 | B_2 | D_1 | D_2 | Source2 |
| = | | | | |
| $A_1B_1 + A_2B_2$ | | $C_1D_1 + C_2D_2$ | | Result1 |

As another example, Table 10 shows the separate multiplication and accumulation of sets A and B and sets C and D, where each of these sets includes 4 data elements.

TABLE 10

| Multiply-Add Source1, Source2 | | | | |
|-------------------------------------|-------|-------------------------------------|-------|---------|
| A_1 | A_2 | C_1 | C_2 | Source1 |
| B_1 | B_2 | D_1 | D_2 | Source2 |
| = | | | | |
| $A_1B_1 + A_2B_2$ | | $C_1D_1 + C_2D_2$ | | Result1 |
| Multiply-Add Source3, Source4 | | | | |
| A_3 | A_4 | C_3 | C_4 | Source3 |
| B_3 | B_4 | D_3 | D_4 | Source4 |
| = | | | | |
| $A_3B_3 + A_4B_4$ | | $C_3D_3 + C_4D_4$ | | Result2 |
| Packed Add Result1, Result2 | | | | |
| $A_1B_1 + A_2B_2$ | | $C_1D_1 + C_2D_2$ | | Result1 |
| $A_3B_3 + A_4B_4$ | | $C_3D_3 + C_4D_4$ | | Result2 |
| = | | | | |
| $A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4$ | | $C_1D_1 + C_2D_2 + C_3D_3 + C_4D_4$ | | Result6 |

3) Dot Product Algorithms

Dot product (also termed as inner product) is used in signal processing and matrix operations. For example, dot product is used when computing the product of matrices, digital filtering operations (such as FIR and IIR filtering), and computing correlation sequences. Since many speech compression algorithms (e.g., GSM, G.728, CELP, and VSELP) and Hi-Fi compression algorithms (e.g., MPEG and subband coding) make extensive use of digital filtering and correlation computations, increasing the performance of dot product increases the performance of these algorithms.

The dot product of two length N sequences A and B is defined as:

$$\text{Result} = \sum_{i=0}^{N-1} A_i \cdot B_i$$

Performing a dot product calculation makes extensive use of the multiply accumulate operation where corresponding

18

elements of each of the sequences are multiplied together, and the results are accumulated to form the dot product result.

The dot product calculation can be performed using the multiply-add instruction. For example if the packed data type containing four sixteen-bit elements is used, the dot product calculation may be performed on two sequences each containing four values by:

- 1) accessing the four sixteen-bit values from the A sequence to generate Source1 using a move instruction;
- 2) accessing four sixteen-bit values from the B sequence to generate Source2 using a move instruction; and
- 3) performing multiplying and accumulating as previously described using a multiply-add, packed add, and shift instructions.

For vectors with more than just a few elements the method shown in Table 10 is used and the final results are added together at the end. Other supporting instructions include the packed OR and XOR instructions for initializing the accumulator register, the packed shift instruction for shifting off unwanted values at the final stage of computation. Loop control operations are accomplished using instructions already existing in the instruction set of processor 109.

4) Discrete Cosign Transform Algorithms

Discrete Cosine Transform (DCT) is a well known function used in many signal processing algorithms. Video and image compression algorithms, in particular, make extensive use of this transform.

In image and video compression algorithms, DCT is used to transform a block of pixels from the spatial representation to the frequency representation. In the frequency representation, the picture information is divided into frequency components, some of which are more important than others. The compression algorithm selectively quantizes or discards the frequency components that do not adversely affect the reconstructed picture contents. In this manner, compression is achieved.

There are many implementations of the DCT, the most popular being some kind of fast transform method modeled based on the Fast Fourier Transform (FFT) computation flow. In the fast transform, an order N transform is broken down to a combination of order N/2 transforms and the result recombined. This decomposition can be carried out until the smallest order 2 transform is reached. This elementary 2 transform kernel is often referred to as the butterfly operation. The butterfly operation is expressed as follows:

$$X = a * x + b * y$$

$$Y = c * x - d * y$$

where a, b, c and d are termed the coefficients, x and y are the input data, and X and Y are the transform output.

The multiply-add allows the DCT calculation to be performed using packed data in the following manner:

- 1) accessing the two 16-bit values representing x and y to generate Source1 (see Table 11 below) using the move and unpack instructions;
- 2) generating Source2 as shown in Table 11 below—Note that Source2 may be reused over a number of butterfly operations; and
- 3) performing a multiply-add instruction using Source1 and Source2 to generate the Result (see Table 11 below).

US 6,385,634 B1

19

TABLE 11

| | | | | |
|-----------|---|-----------|----|---------|
| x | y | x | y | Source1 |
| a | b | c | -d | Source2 |
| a*x + b*y | | c*x - d*y | | Source3 |

In some situations, the coefficients of the butterfly operation are 1. For these cases, the butterfly operation degenerates into just adds and subtracts that may be performed using the packed add and packed subtract instructions.

An IEEE document specifies the accuracy with which inverse DCT should be performed for video conferencing. (See, IEEE Circuits and Systems Society, "IEEE Standard Specifications for the Implementations of 8x8 Inverse Discrete Cosine Transform," IEEE Std. 1180-1990, IEEE Inc. 345 East 47th St., NY, N.Y. 10017, USA, Mar. 18, 1991). The required accuracy is met by the disclosed multiply-add instruction because it uses 16-bit inputs to generate 32-bit outputs.

In this manner, the described multiply-add instruction can be used to improve the performance of a number of different algorithms, including algorithms that require the multiplication of complex numbers, algorithms that require transforms, and algorithms that require multiply accumulate operations. As a result, this multiply-add instruction can be used in a general purpose processor to improve the performance of a greater number algorithms than the described prior art instructions.

Alternative Embodiments

While the described embodiment uses 16-bit data elements to generate 32-bit data elements, alternative embodiments could use different sized inputs to generate different sized outputs. In addition, while in the described embodiment Source1 and Source2 each contain 4 data elements and the multiply-add instruction performs two multiply-add operations, alternative embodiment could operate on packed data having more or less data elements. For example, one alternative embodiment operates on packed data having 8 data elements using 4 multiply-adds generating a resulting packed data having 4 data elements. While in the described embodiment each multiply-add operation operates on 4 data elements by performing 2 multiplies and 1 addition, alternative embodiments could be implemented to operate on more or less data elements using more or less multiplies and additions. As an example, one alternative embodiment operates on 8 data elements using 4 multiplies (one for each pair of data elements) and 3 additions (2 additions to add the results of the 4 multiplies and 1 addition to add the results of the 2 previous additions).

While the invention has been described in terms of several embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described. The method and apparatus of the invention can be practiced with modification and alteration within the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of limiting on the invention.

What is claimed is:

1. A computer-implemented method comprising:
responsive to the execution of a single instruction that specifies a first and second packed data including data elements, wherein said data elements of said first

20

packed data comprise a first, second, third, and fourth value, and wherein said data elements of said second packed data comprise a fourth, fifth, sixth, seventh, and eighth value, performing,

- A) multiplying together said first value and said second value to generate a first intermediate result;
- B) multiplying together said third value and said fourth value to generate a second intermediate result;
- C) multiplying together said fifth value and said sixth value to generate a third intermediate result;
- D) multiplying together said seventh value and said eighth value to generate a fourth intermediate result;
- E) adding together said first intermediate result and said second intermediate result to generate a fifth intermediate result;
- F) adding together said third intermediate result and said fourth intermediate result to generate a sixth intermediate result;
- G) storing said fifth intermediate result as a first data element of a third packed data and said sixth intermediate result as a second data element of said third packed data, wherein said third packed data includes only said first and second data elements, and said first and second data elements have twice as many bits as the data elements in said first and second packed data, and
- H) completing execution of said single instruction without adding said first and second data elements of said third packed data.

2. The method of claim 1 further including:

accessing said first, third, fifth, and seventh values from a storage area; and writing said first packed data over said first, third, fifth, and seventh values in said storage area.

3. In a computer system, a method comprising:

responsive to the execution of a single instruction that specifies a first packed data and a second packed data, said first packed data including A_1, A_2, A_3, A_4 as data elements, said second packed data including $B_1, B_2, B_3,$ and B_4 as data elements, performing
performing the operation $(A_1 \times B_1) + (A_2 \times B_2)$ to generate a first data element in a third packed data;
performing the operation $(A_3 \times B_3) + (A_4 \times B_4)$ to generate a second data element in said third packed data;
completing execution of said single instruction without adding said first and second data elements of said third packed data; and
storing said third packed data for use as an operand to another instruction.

4. The method of claim 3, further including:

accessing said first packed data from a register; and writing said third packed data over said first packed data in said register.

5. In a computer system having stored therein a first packed data and a second packed data each containing initial data elements, each of said initial data elements in said first packed data having a corresponding initial data element in said second packed data, a method for performing multiply add operations in response to a single instruction, said method comprising the steps of:

multiplying together said corresponding initial data elements in said first packed data and said second packed data to generate corresponding intermediate data elements, said intermediate data elements being divided into a number of sets;

generating a plurality of result data elements, a first of said plurality of result data elements representing the

US 6,385,634 B1

21

sum of said intermediate result data elements in a first of said number of sets, a second of said plurality of result data elements representing the sum of said intermediate result data elements in a second of said number of sets; and

completing execution of said single instruction without summing said plurality of result data elements.

6. The method of claim 5, further including:

storing said plurality of result data elements as a third packed data for use as an operand to another instruction.

7. The method of claim 5 further including:

accessing said first and second packed data from a register; and

writing said plurality of result data elements over said first packed data in said register.

8. In a computer system, a method comprising:

responsive to the execution of a single instruction that specifies a first packed data including A_1 , A_2 , A_3 , and A_4 as data elements, and a second packed data including B_1 , B_2 , B_3 , and B_4 as data elements, performing,

multiplying together A_1 and B_1 to generate a first intermediate result;

multiplying together A_2 and B_2 to generate a second intermediate result;

multiplying together A_3 and B_3 to generate a third intermediate result;

multiplying together A_4 and B_4 to generate a fourth intermediate result;

performing in parallel the following steps:

adding together said first intermediate result and said second intermediate result to generate a first data element in a third packed data; and

adding together said third intermediate result and said fourth intermediate result to generate a second data element in said third packed data; and

saving said third packed data for use as an operand to another instruction.

9. The method of claim 8, further including:

accessing said first and second packed data from a storage area; and

writing third packed data in said storage area.

10. In a computer system, a method comprising:

responsive to the execution of a single instruction that specifies a first packed data and a second packed data each containing initial data elements, each of said initial data elements in said first packed data having a corresponding initial data element in said second packed data, performing

multiplying together said corresponding initial data elements in said first packed data and said second packed data to generate corresponding intermediate data elements, said intermediate data elements being divided into a number of sets;

generating a plurality of result data elements, a first of said plurality of result data elements representing the sum of said intermediate result data elements in a first

22

of said number of sets, a second of said plurality of result data elements representing the sum of said intermediate result data elements in a second of said number of sets; and

completing execution of said single instruction without summing said plurality of result data elements.

11. The method of claim 1 wherein said first and second packed data are specified by including register designations in said single instruction.

12. A computer-implemented method comprising:

responsive to the execution of a single instruction that specifies a first and second storage areas having respectively stored therein a first and second packed data, said first and second packed data each having a first, second, third, and fourth data elements, performing,

multiplying together said first data elements to generate a first intermediate result,

multiplying together said second data elements to generate a second intermediate result,

multiplying together said third data elements to generate a third intermediate result,

multiplying together said fourth data elements to generate a fourth intermediate result,

adding together said first intermediate result and said second intermediate result to generate a fifth intermediate result,

adding together said third intermediate result and said fourth intermediate result to generate a sixth intermediate result, and

storing said fifth and sixth intermediate results as first and second unaccumulated data elements of a third packed data, respectively, wherein said third packed data includes only said first and second data elements.

13. The method of claim 12, wherein said first and second unaccumulated data elements of said third packed data contain twice as many bits as the data elements in said first and second packed data.

14. In a computer system, a method comprising:

responsive to the execution of a single instruction, performing,

fetching a first packed data and a second packed data, said first packed data including A_1 , A_2 , A_3 , and A_4 as data elements, said second packed data including B_1 , B_2 , B_3 , B_4 , as data elements;

performing the operation $(A_1 \times B_1) + (A_2 \times B_2)$ to generate a first result;

performing the operation $(A_3 \times B_3) + (A_4 \times B_4)$ to generate a second result;

storing said first and second results as unaccumulated data elements in a third storage area, said third storage area having at least a first field and a second field, said first field for saving said first result as a first unaccumulated data element of said third storage area, and said second field for saving said second result as a second unaccumulated data element of said third storage area.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,385,634 B1
DATED : May 7, 2002
INVENTOR(S) : Peleg et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 15,

Line 32, delete " i_2 ", insert -- i_1 --.

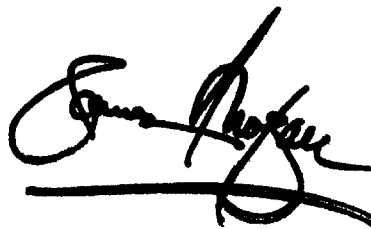
Line 33, delete " i_1 ", insert -- i_2 --.

Line 44, delete " i_2 ", insert -- i_1 --.

Line 45, delete " $-i_1$ ", insert -- $-i_2$ --.

Signed and Sealed this

Seventh Day of January, 2003

A handwritten signature in black ink, appearing to read "James E. Rogan", with a long horizontal flourish underneath.

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

EXHIBIT 7



US006418529B1

(12) **United States Patent**
Roussel

(10) **Patent No.:** **US 6,418,529 B1**
(45) **Date of Patent:** ***Jul. 9, 2002**

(54) **APPARATUS AND METHOD FOR
PERFORMING INTRA-ADD OPERATION**

(75) Inventor: **Patrice Roussel**, Portland, OR (US)

(73) Assignee: **Intel Corporation**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/053,401**

(22) Filed: **Mar. 31, 1998**

(51) **Int. Cl.**⁷ **G06F 9/302**

(52) **U.S. Cl.** **712/221; 712/222**

(58) **Field of Search** **712/42, 208, 221; 708/490**

(56) **References Cited**

U.S. PATENT DOCUMENTS

| | | | |
|-------------|---------|--------------------|---------|
| 3,711,692 A | 1/1973 | Batcher | 708/210 |
| 3,723,715 A | 3/1973 | Chen et al. | 708/709 |
| 4,161,784 A | 7/1979 | Cushing et al. | 708/513 |
| 4,189,716 A | 2/1980 | Krambeck | 341/63 |
| 4,393,468 A | 7/1983 | New | 708/518 |
| 4,418,383 A | 11/1983 | Doyle et al. | 710/127 |
| 4,498,177 A | 2/1985 | Larson | 714/806 |
| 4,630,192 A | 12/1986 | Wassel et al. | 712/221 |
| 4,707,800 A | 11/1987 | Montrone et al. | 708/714 |
| 4,771,379 A | 9/1988 | Ando et al. | 712/42 |
| 4,785,393 A | 11/1988 | Chu et al. | 712/221 |
| 4,785,421 A | 11/1988 | Takahashi et al. | 708/205 |
| 4,901,270 A | 2/1990 | Galbi et al. | 708/708 |
| 4,989,168 A | 1/1991 | Kuroda et al. | 708/210 |
| 5,095,457 A | 3/1992 | Jeong | 708/626 |
| 5,187,679 A | 2/1993 | Vassiliadis et al. | 708/706 |
| 5,201,056 A | 4/1993 | Daniel et al. | 712/41 |
| 5,327,369 A | 7/1994 | Ashkenazi | |
| 5,339,447 A | 8/1994 | Balmer | 377/82 |
| 5,390,135 A | 2/1995 | Lee et al. | |

| | | | |
|-------------|---------|-----------------|---------|
| 5,418,736 A | 5/1995 | Widigen et al. | 708/706 |
| 5,442,799 A | 8/1995 | Murakami et al. | 712/36 |
| 5,448,703 A | 9/1995 | Amini et al. | 710/110 |
| 5,517,626 A | 5/1996 | Archer et al. | 710/11 |
| 5,530,661 A | 6/1996 | Garbe et al. | 708/420 |
| 5,537,601 A | 7/1996 | Kimura et al. | 712/35 |
| 5,586,070 A | 12/1996 | Purcell | 708/620 |

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

| | | |
|----|--------------|---------|
| EP | 0 318 957 A3 | 11/1988 |
| WO | WO 97/23821 | 7/1997 |

OTHER PUBLICATIONS

MIPS Extension for Digital Media with 3D, MIPS Technology, Inc., Mar. 12, 1997, pp 0-26.

A Processor Architecture for 3D Graphics Calculations, Yulun Wang, Amante Manager, Partha Srinivasan, Computer Motion, Inc., pp 1-23.

Parallel Computers for Graphics Applications (Proceedings: Second International Conference . . .), Levinthal, et al., 1987, pp 193-198.

(List continued on next page.)

Primary Examiner—David Y. Eng

(74) *Attorney, Agent, or Firm*—Blakely, Sokoloff, Taylor & Zafman LLP

(57) **ABSTRACT**

Method and apparatus for including in a processor, instructions for performing intra-add operations on packed data. In one embodiment, an execution unit is coupled to a storage area. The storage area has stored therein a first packed data operand and a second packed data operand. The execution unit performs operations on data elements in the first packed data operand and the second packed data operand to generate a plurality of data elements in a packed data result in response to receiving a single instruction. At least two of the plurality of data elements in a packed data result store the result of an intra-add operation using the first packed data operand and the second packed data operand.

58 Claims, 13 Drawing Sheets

INTRA-ADD: IRESULT1-IRESULT1; I RESULT2-IRESULT2

| | | | | | | | | |
|-------|----|-------|----|-------|----|-------|---|----------|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 | |
| A14X4 | | A13X3 | | A12X2 | | A11X1 | | IResult1 |

| | | | | | | | | |
|-------|----|-------|----|-------|----|-------|---|----------|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 | |
| A24X4 | | A23X3 | | A22X2 | | A21X1 | | IResult2 |

| | | | | | | | | |
|-------------|----|-------------|----|-------------|----|-------------|---|--|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 | |
| A24X4+A23X3 | | A22X2+A21X1 | | A14X4+A13X3 | | A12X2+A11X1 | | |

IAResult1

US 6,418,529 B1

Page 2

U.S. PATENT DOCUMENTS

| | | | |
|---------------|---------|-------------------|------------|
| 5,677,862 A | 10/1997 | Peleg et al. | |
| 5,678,009 A | 10/1997 | Bains et al. | 710/125 |
| 5,721,697 A | 2/1998 | Lee | 708/620 |
| 5,721,892 A | 2/1998 | Peleg et al. | |
| 5,815,421 A | 9/1998 | Dulong et al. | |
| 5,819,117 A | 10/1998 | Hansen | |
| 5,822,232 A | 10/1998 | Dulong et al. | |
| 5,859,790 A * | 1/1999 | Sidwell | 708/607 |
| 5,862,067 A | 1/1999 | Mennemeier et al. | |
| 5,875,355 A * | 2/1999 | Mackenzie et al. | 708/607 |
| 5,880,984 A | 3/1999 | Burchfiel et al. | 708/501 |
| 5,880,985 A | 3/1999 | Makineni et al. | 708/625 |
| 5,883,824 A | 3/1999 | Lee et al. | |
| 5,887,186 A * | 3/1999 | Nakanishi | 712/28 |
| 5,901,301 A | 5/1999 | Matsuo et al. | 712/212 |
| 5,918,062 A * | 6/1999 | Oberman et al. | 395/800.07 |
| 5,983,257 A | 11/1999 | Dulong et al. | 708/203 |
| 6,006,316 A | 12/1999 | Dinkjian | |
| 6,014,684 A * | 1/2000 | Hoffman | 708/620 |
| 6,014,735 A | 1/2000 | Chennupaty et al. | |
| 6,041,404 A * | 3/2000 | Roussel et al. | 712/210 |
| 6,115,812 A | 9/2000 | Abdallah et al. | |
| 6,122,725 A | 9/2000 | Roussel et al. | |
| 6,211,892 B1 | 4/2001 | Huff et al. | |
| 6,212,618 B1 | 4/2001 | Roussel | |
| 6,230,253 B1 | 5/2001 | Roussel et al. | |
| 6,230,257 B1 | 5/2001 | Roussel et al. | |
| 6,288,723 B1 | 9/2001 | Huff et al. | |

OTHER PUBLICATIONS

A SIMD Graphics Processor, Adam Levinthal, Thomas Porter, 1984, pp 77-82.
 Architecture of a Broadband Mediaprocessor (Proceedings of COMPCON '96), Craig Hansen, 1996, pp 334-354.
 64-bit and Multimedia Extensions in the PA-RISC 2.0 Architecture, Computing Directory Technologies Precision Architecture Document, Jul. 17, 1997.
 Silicon Graphics Introduces Enhanced MIPS Architecture to Lead the Interactive Digital Revolution, Oct. 21, 1996.
 21164 Alpha Microprocessor Data Sheet, Samsung Electronics, 1997.

TM100-Preliminary Data Book, Philips Semiconductors, Jul. 1, 1997, pp A-74, A133-138, A161.

Visual Instruction Set (VIS) User's Guide, Sun Microsystems, Version 1.1, Mar. 1997, pp i-xii, 1-127.

AMD-3D Technology Manual, Advance Micro Devices, (AMD), Feb. 1998, pp i-x 1-58.

J. Shipnes, Graphics Processing with the 88110 RISC Microprocessor, IEEE (1992), pp. 169-174.

TMS320C2x User's Guide, Texas Instruments (1993) pp. 3-2 through 3-11; 3-28 through 3-34; 4-1 through 4-22; 4-41; 4-1-3 through 4-120; 4-122; 4-150 through 4-151.

SPARC Technology Business, UltraSPARC Multimedia Capabilities On-Chip Support for Real-Time Video and Advanced Graphics, Sun Microsystems (Sep. 1994).

IBM Technical Disclosure Bulletin, vol. 35, No. 1A, Bit Zone Accumulator, Jun. 1992, p. 106.

Intel i750, i860; i960 Processors and Related Products, 1993, pp 1-3.

MC88110 Programmer's Reference Guide, Motorola Inc. (1992), p 1-4.

MC88110 Second Generation RISC Microprocessor User's Manual, Motorola, Inc. (1991).

Errata to MC88110 Second Generation RISC Microprocessor User's Manual, Motorola, (Sep. 1992).

GWENNAP, New PA-RISC Processor Decodes MPEG Video, Microprocessor Report (Jan. 1994) pp. 16,17.

CASE, Philips Hopes to Displace DSPs with VLIW, Microprocessor Report (Dec. 1994), pp. 12-15.

J.F. Takie, et al., "Comparison of Some Parallel Matrix Multiplication Algorithms", 8th Mediterranean Electrotechnical Conference, Melecon '96, vol. 1, 1996, pp 155-158.

H. Barad, et al., "Intel's Multimedia Architecture Extension", Nineteenth Convention of Electrical and Electronics Engineers in Isreal, 1996, pp. 158-151.

U.S. Pat. Application No. 08/521,360, titled "A set of Instructions for Operating on a Packed Data", filed on Aug. 31, 1995, assigned to Intel Corporation, now abandoned.

* cited by examiner

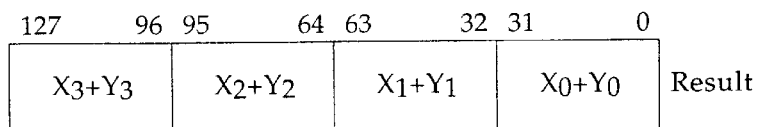
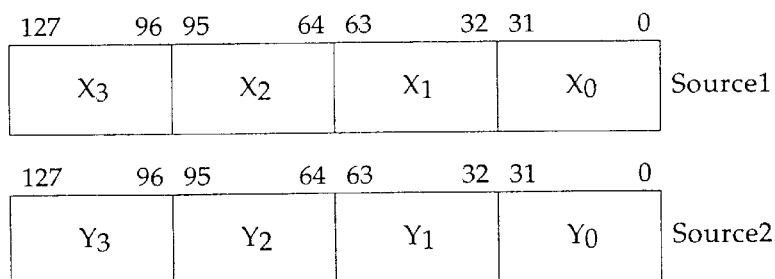
U.S. Patent**Jul. 9, 2002****Sheet 1 of 13****US 6,418,529 B1**

FIGURE 1
(PRIOR ART)

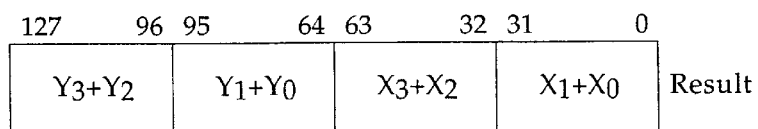
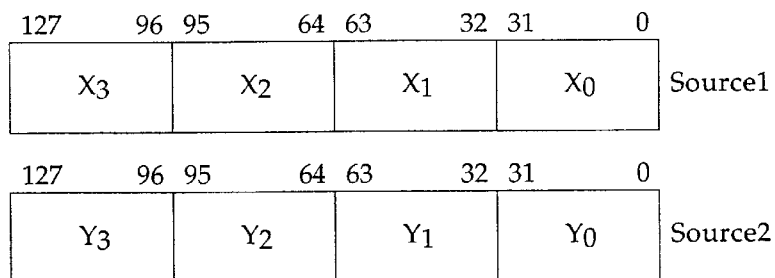
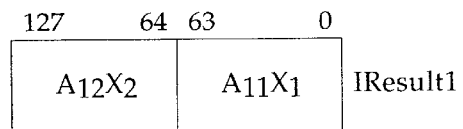
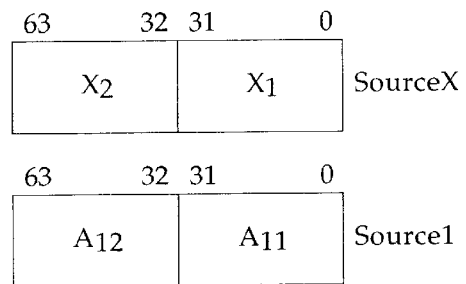
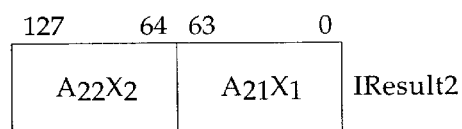
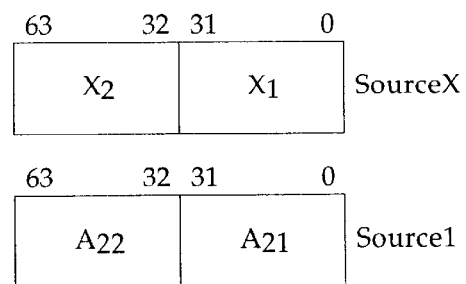


FIGURE 2

U.S. Patent**Jul. 9, 2002****Sheet 2 of 13****US 6,418,529 B1****MULTIPLY - SOURCE 1, SOURCE 2****FIGURE 3A****FIGURE 3B**

U.S. Patent

Jul. 9, 2002

Sheet 3 of 13

US 6,418,529 B1

INTRA-ADD: DATA ELEMENTS OF IRESULT2; IRESULT1

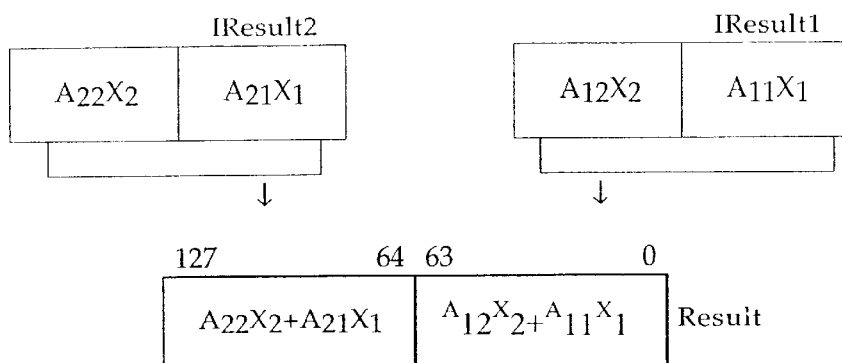


FIGURE 3C

MULTIPLY: SOURCE1, SOURCE2

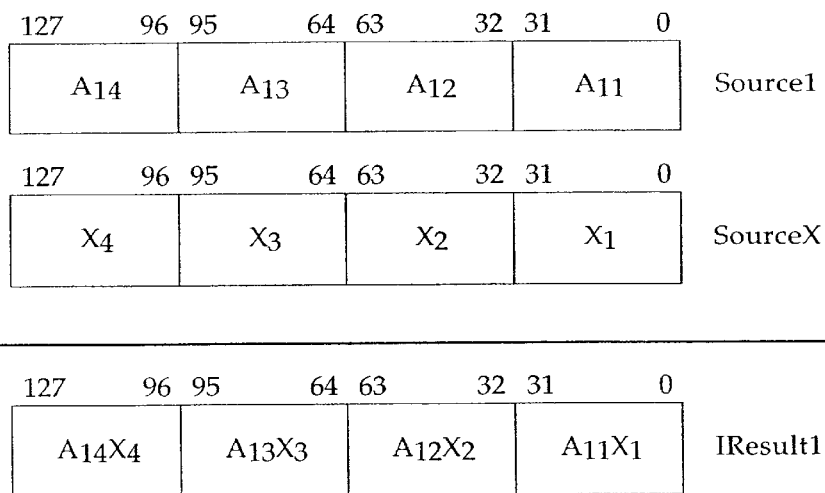
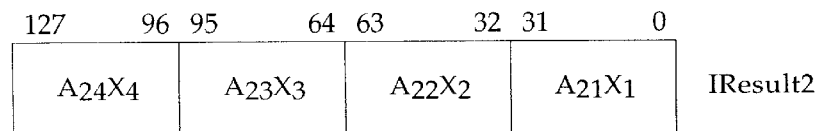
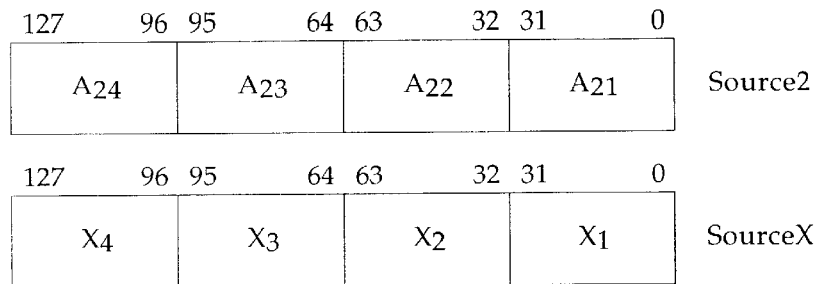
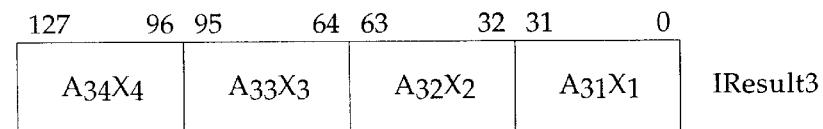
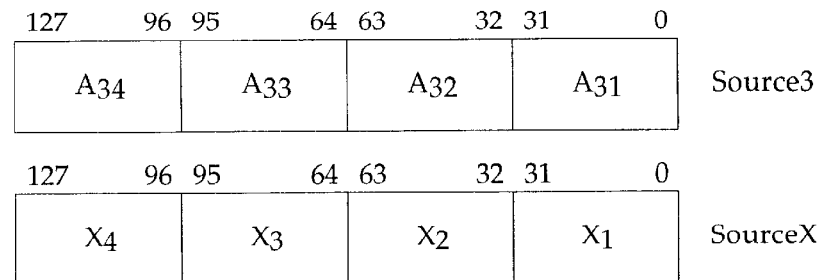
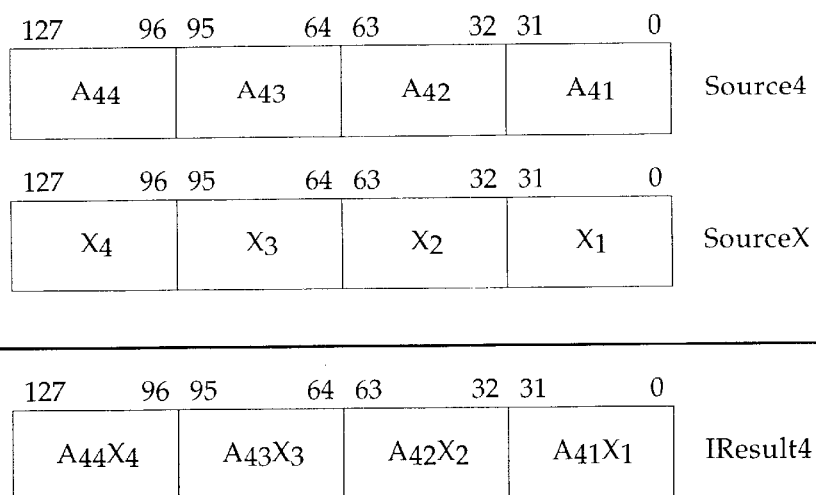


FIGURE 4A

U.S. Patent**Jul. 9, 2002****Sheet 4 of 13****US 6,418,529 B1****FIGURE 4B****FIGURE 4C**

U.S. Patent**Jul. 9, 2002****Sheet 5 of 13****US 6,418,529 B1****FIGURE 4D**

U.S. Patent**Jul. 9, 2002****Sheet 6 of 13****US 6,418,529 B1****INTRA-ADD: IRESULT1-IRESULT1; I RESULT2-IRESULT2**

| | | | | | | | | |
|--------------------------------|----|--------------------------------|----|--------------------------------|----|--------------------------------|---|----------|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 | |
| A ₁₄ X ₄ | | A ₁₃ X ₃ | | A ₁₂ X ₂ | | A ₁₁ X ₁ | | IResult1 |

| | | | | | | | | |
|--------------------------------|----|--------------------------------|----|--------------------------------|----|--------------------------------|---|----------|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 | |
| A ₂₄ X ₄ | | A ₂₃ X ₃ | | A ₂₂ X ₂ | | A ₂₁ X ₁ | | IResult2 |

| | | | | | | | |
|--|----|--|----|--|----|--|---|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 |
| A ₂₄ X ₄ +A ₂₃ X ₃ | | A ₂₂ X ₂ +A ₂₁ X ₁ | | A ₁₄ X ₄ +A ₁₃ X ₃ | | A ₁₂ X ₂ +A ₁₁ X ₁ | |

IAResult1

FIGURE 4E

U.S. Patent**Jul. 9, 2002****Sheet 7 of 13****US 6,418,529 B1****INTRA-ADD: IRESULT3-IRESULT3; I RESULT4-IRESULT4**

| | | | | | | | | |
|--------------------------------|----|----|----|--------------------------------|----|--------------------------------|---|---------------------------------|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 | |
| A ₃₄ X ₄ | | | | A ₃₃ X ₃ | | A ₃₂ X ₂ | | A ₃₁ X ₁ |
| IResult3 | | | | | | | | |
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 | |
| A ₄₄ X ₄ | | | | A ₄₃ X ₃ | | A ₄₂ X ₂ | | aA ₄₁ X ₁ |
| IResult4 | | | | | | | | |

| | | | | | | | |
|--|----|----|--|----|--|----|--|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 |
| A ₄₄ X ₄ +A ₄₃ X ₃ | | | A ₄₂ X ₂ +A ₄₁ X ₁ | | A ₃₄ X ₄ +A ₃₃ X ₃ | | A ₃₂ X ₂ +A ₃₁ X ₁ |

IAResult2

FIGURE 4F

U.S. Patent**Jul. 9, 2002****Sheet 8 of 13****US 6,418,529 B1****INTRA-ADD**

| | | | | | | | |
|-------------------------|----|-------------------------|----|-------------------------|----|-------------------------|---|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 |
| $A_{24}X_4 + A_{23}X_3$ | | $A_{22}X_2 + A_{21}X_1$ | | $A_{14}X_4 + A_{13}X_3$ | | $A_{12}X_2 + A_{11}X_1$ | |

IAResult1

| | | | | | | | |
|-------------------------|----|-------------------------|----|-------------------------|----|-------------------------|---|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 |
| $A_{44}X_4 + A_{43}X_3$ | | $A_{42}X_2 + A_{41}X_1$ | | $A_{34}X_4 + A_{33}X_3$ | | $A_{32}X_2 + A_{31}X_1$ | |

IAResult2

| | | | | | | | |
|--|----|--|----|--|----|--|---|
| 127 | 96 | 95 | 64 | 63 | 32 | 31 | 0 |
| $A_{44}X_4 + A_{43}X_3 +$ $A_{42}X_2 + A_{41}X_1$ | | $A_{34}X_4 + A_{33}X_3 +$ $A_{32}X_2 + A_{31}X_1$ | | $A_{24}X_4 + A_{23}X_3 +$ $A_{22}X_2 + A_{21}X_1$ | | $A_{14}X_4 + A_{13}X_3 +$ $A_{12}X_2 + A_{11}X_1$ | |

Result

FIGURE 4G

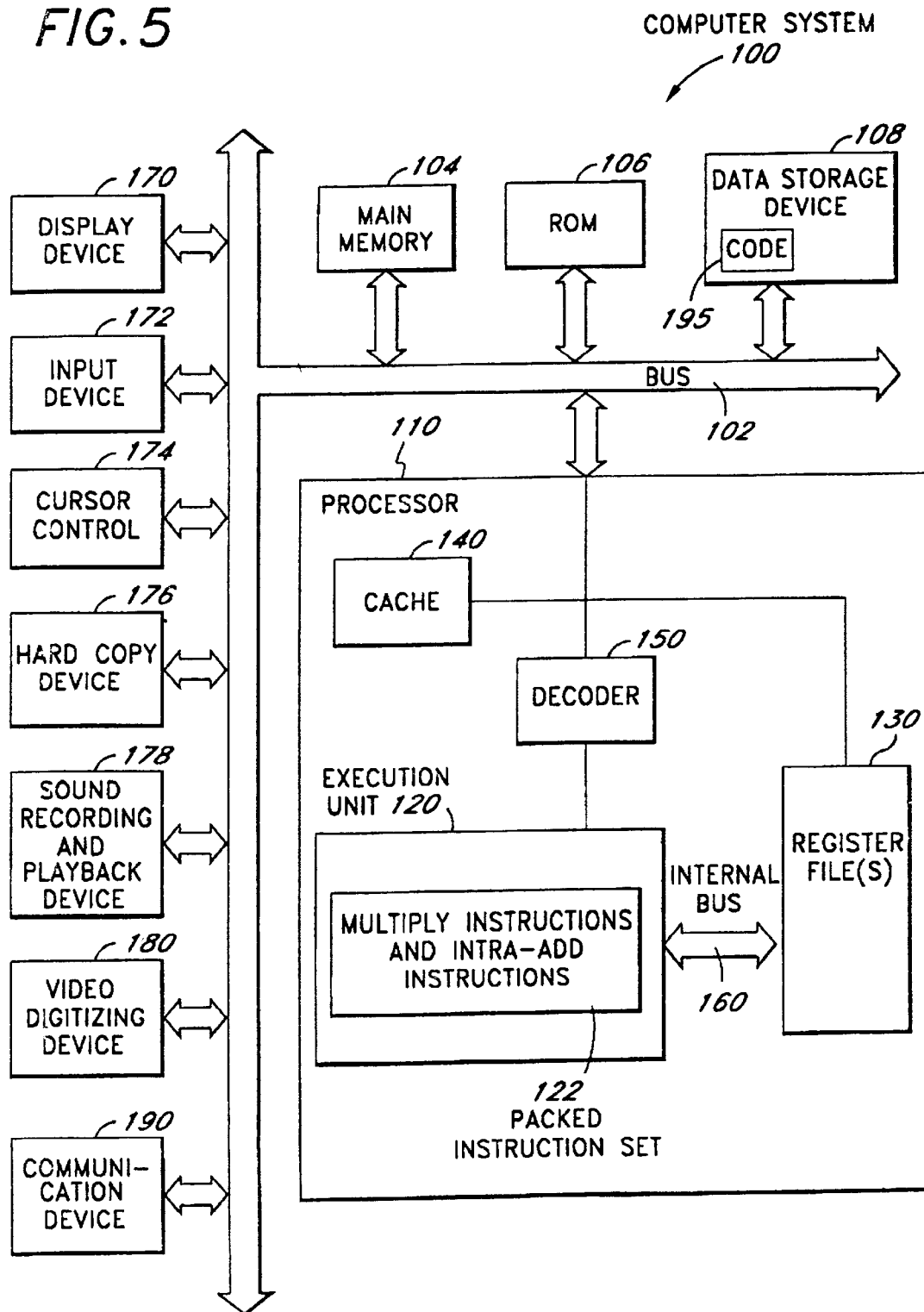
U.S. Patent

Jul. 9, 2002

Sheet 9 of 13

US 6,418,529 B1

FIG. 5

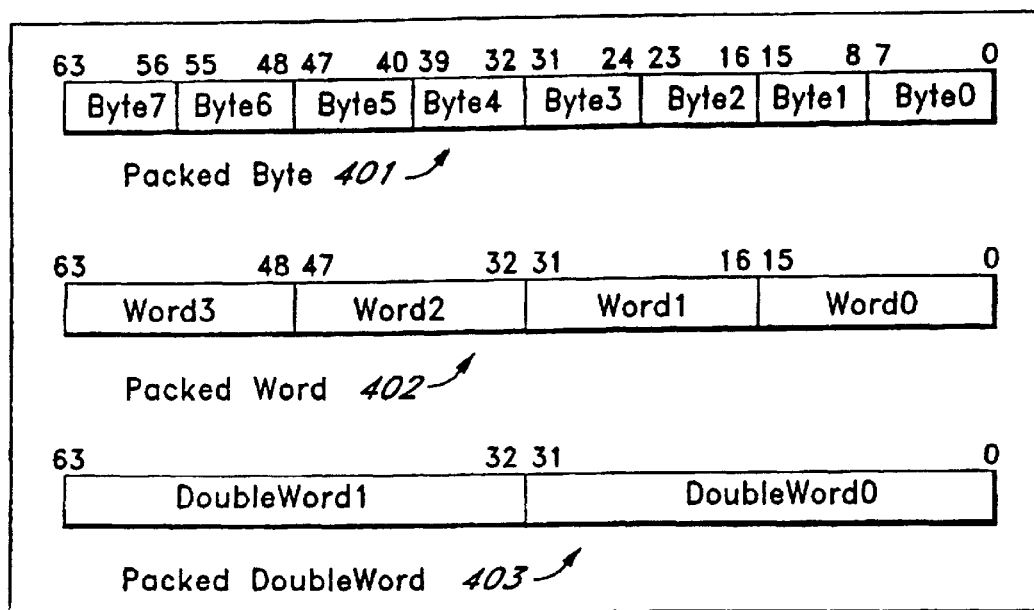
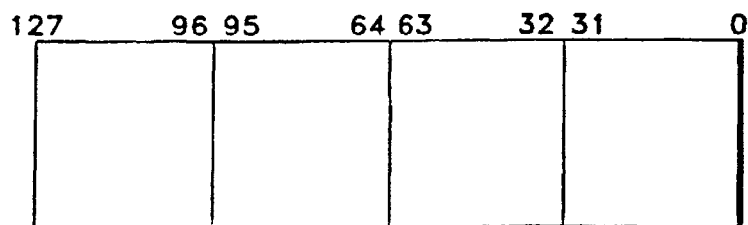


U.S. Patent

Jul. 9, 2002

Sheet 10 of 13

US 6,418,529 B1

FIG. 6**FIG. 7**

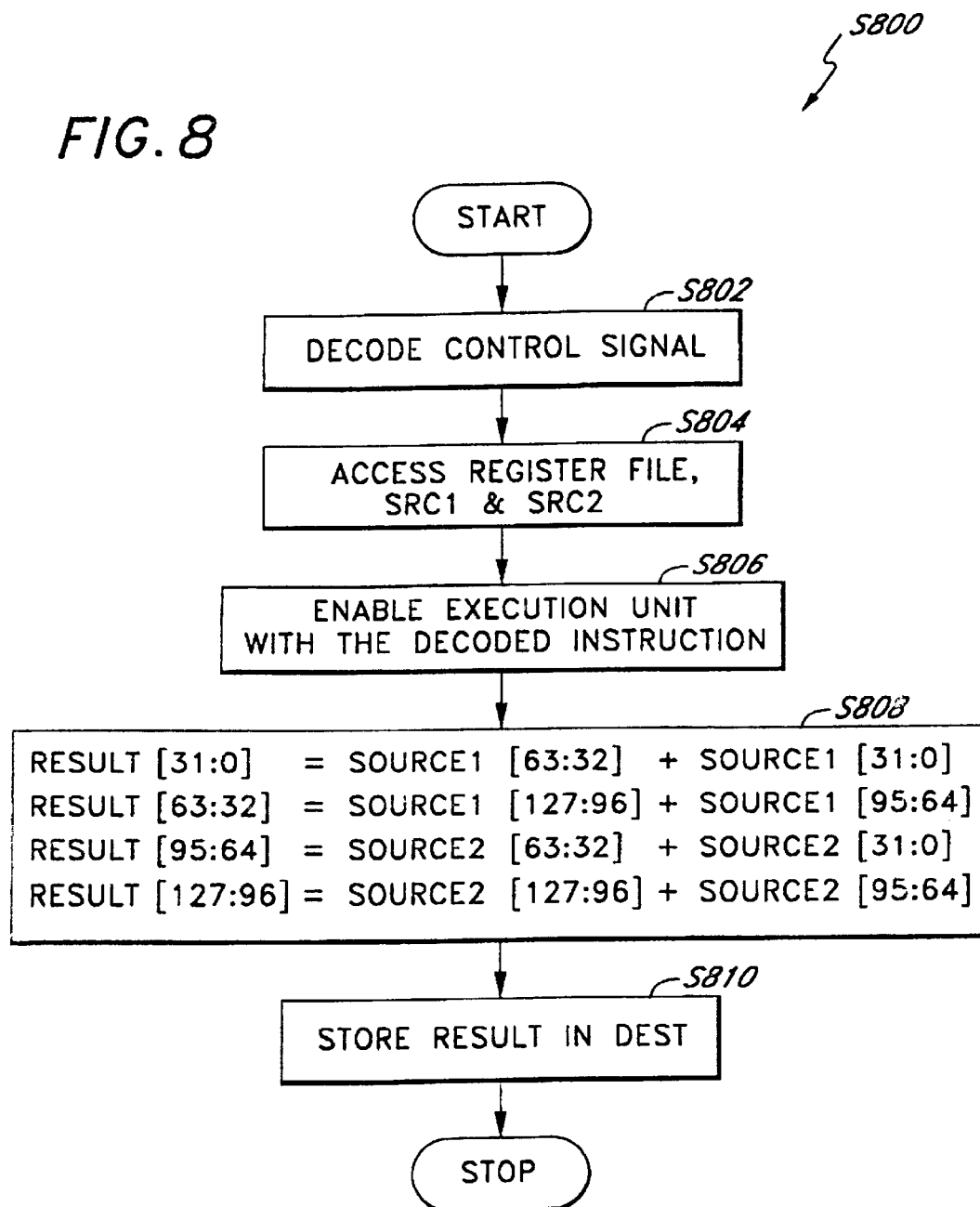
U.S. Patent

Jul. 9, 2002

Sheet 11 of 13

US 6,418,529 B1

FIG. 8



U.S. Patent

Jul. 9, 2002

Sheet 12 of 13

US 6,418,529 B1

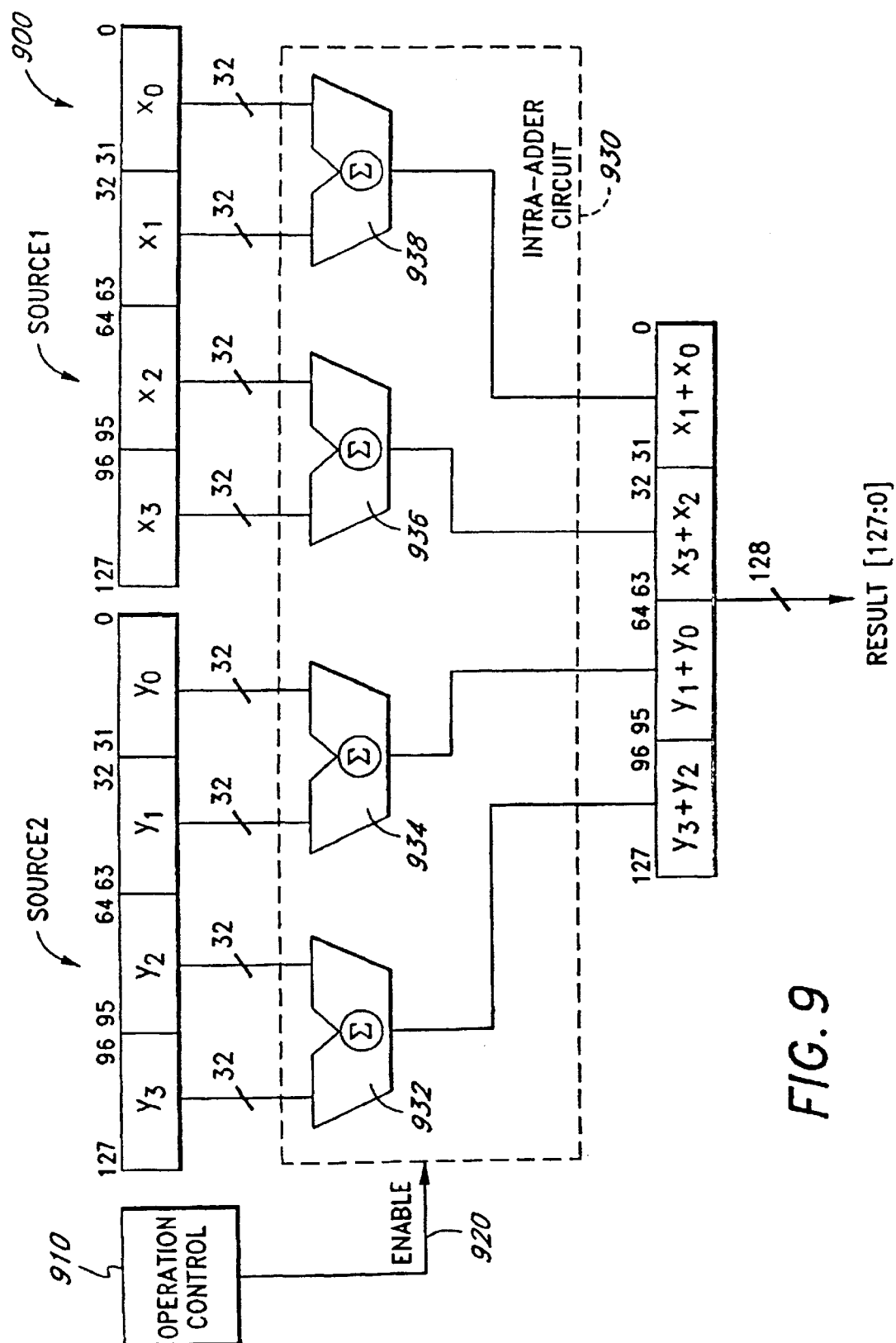


FIG. 9

U.S. Patent

Jul. 9, 2002

Sheet 13 of 13

US 6,418,529 B1

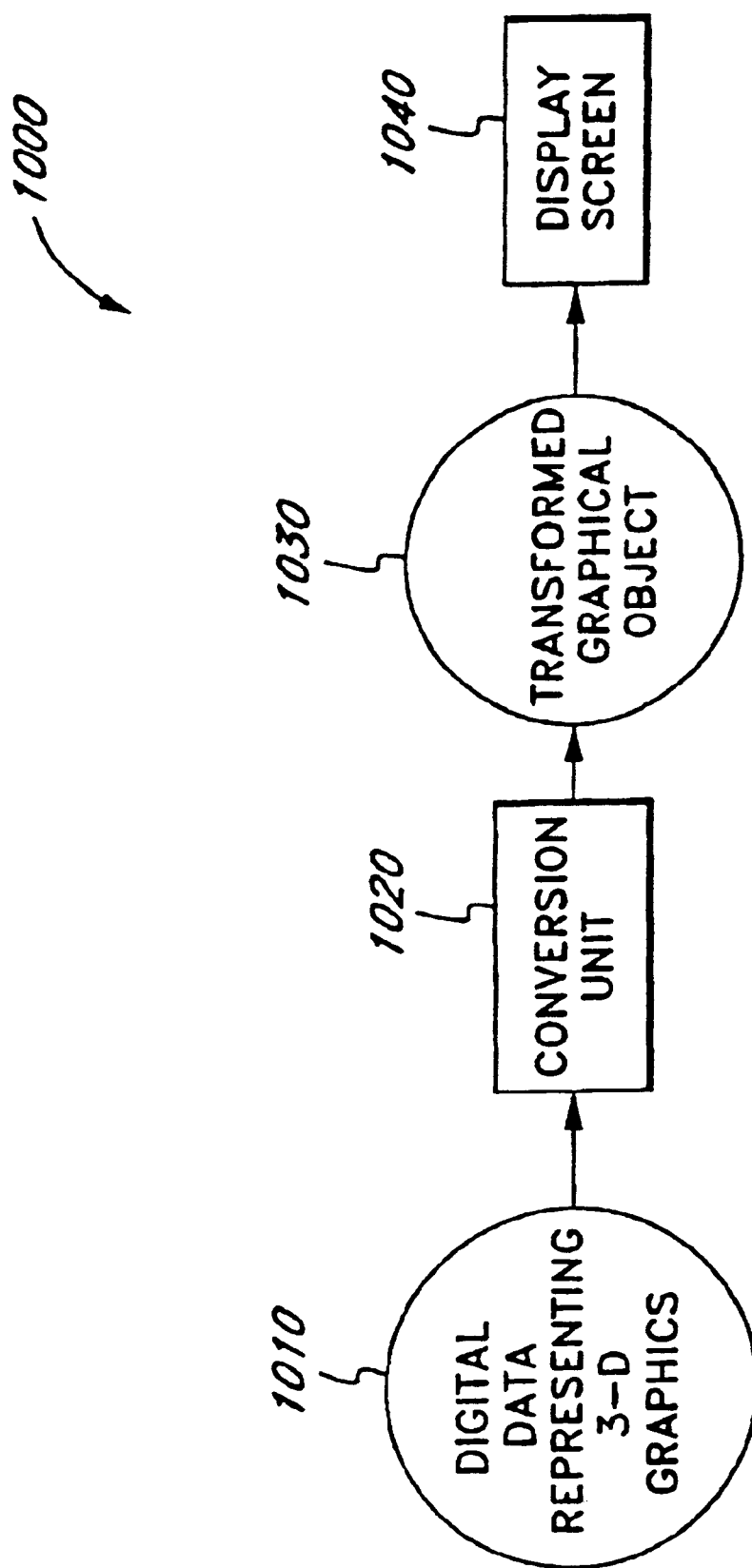


FIG. 10

US 6,418,529 B1

1

APPARATUS AND METHOD FOR
PERFORMING INTRA-ADD OPERATIONCROSS REFERENCE TO RELATED
APPLICATIONS

This application is related to U.S. patent application Ser. No. 09/053,388, entitled "APPARATUS AND METHOD FOR PERFORMING MULTI-DIMENSIONAL COMPUTATIONS BASED ON INTRA-ADD OPERATION", filed Mar. 31, 1998 by Patrice Roussel, now U.S. Pat. No. 6,212,618; and, this application is also related to U.S. patent application Ser. No. 09/053,389, entitled "SYSTEM AND METHOD FOR PERFORMING AN INTRA-ADD OPERATION", filed Mar. 31, 1998 by Thomas Huff and Shreekanth Thakkar, now U.S. Pat. No. 6,211,892.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates in general to the field of computer systems, and in particular, to an apparatus and method for performing multi-dimensional computations based on an intra-add operation.

2. Description of the Related Art

To improve the efficiency of multimedia applications, as well as other applications with similar characteristics, a Single Instruction, Multiple Data (SIMD) architecture has been implemented in computer systems to enable one instruction to operate on several operands simultaneously, rather than on a single operand. In particular, SIMD architectures take advantage of packing many data elements within one register or memory location. With parallel hardware execution, multiple operations can be performed on separate data elements with one instruction, resulting in significant performance improvement.

Currently, the SIMD addition operation only performs "vertical" or inter-register addition, where pairs of data elements, for example, a first element X_n (where n is an integer) from one operand, and a second element Y_n from a second operand, are added together. An example of such a vertical addition operation is shown in FIG. 1, where the instruction is performed on the sets of data elements (X_3, X_2, X_1 and X_0) and (Y_3, Y_2, Y_1 , and Y_0) accessed as Source1 and Source2, respectively to obtain the result ($X_3+Y_3, X_2+Y_2, X_1+Y_1$, and X_0+Y_0).

2

Although many applications currently in use can take advantage of such a vertical add operation, there are a number of important applications which would require the rearrangement of the data elements before the vertical add operation can be implemented so as to provide realization of the application.

For example, a matrix multiplication operation is shown below.

MATRIX A * VECTOR X = VECTOR Y

$$\begin{bmatrix} A_{14} & A_{13} & A_{12} & A_{11} \\ A_{24} & A_{23} & A_{22} & A_{21} \\ A_{34} & A_{33} & A_{32} & A_{31} \\ A_{44} & A_{43} & A_{42} & A_{41} \end{bmatrix} \times \begin{bmatrix} X_4 \\ X_3 \\ X_2 \\ X_1 \end{bmatrix} = \begin{bmatrix} A_{14}X_4 + A_{13}X_3 + A_{12}X_2 + A_{11}X_1 \\ A_{24}X_4 + A_{23}X_3 + A_{22}X_2 + A_{21}X_1 \\ A_{34}X_4 + A_{33}X_3 + A_{32}X_2 + A_{31}X_1 \\ A_{44}X_4 + A_{43}X_3 + A_{42}X_2 + A_{41}X_1 \end{bmatrix}$$

To obtain the product of the matrix A with a vector X to obtain the resulting vector Y, instructions are used to: 1) store the columns of the matrix A as packed operands (this typically requires rearrangement of data because the rows of the matrix A coefficients are stored to be accessed as packed data operands, not the columns); 2) store a set of operands that each have a different one of the vector X coefficients in every data element; 3) use vertical multiplication where each data element in the vector X (i.e., X_4, X_3, X_2, X_1) has to be first multiplied with data elements in each column (for example, [$A_{14}, A_{24}, A_{34}, A_{44}$]) of the matrix A. The results of the multiplication operations are then added together through three vertical add operations such as that shown in FIG. 1, to obtain the final result. Such a matrix multiplication operation based on the use of vertical add operations typically requires 20 instructions to implement, an example of which is shown below in Table 1.

Exemplary Code Based on Vertical-Add Operations

Assumptions:

- 1/X stored with X1 first, X4 last
- 2/transposed of A stored with A11 first, A21 second, A31 third, etc.
- 3/availability of:
 - DUPLS: duplicate once
 - DUPLD: duplicate twice

TABLE 1

| | | |
|-------|-------------------|---|
| MOVD | mm0, <mem_X> | // [0, 0, 0, X1] |
| DUPLS | mm0, mm0 | // [0, 0, X1, X1] |
| DUPLD | mm0, mm0 | // [X1, X1, X1, X1] |
| PFMUL | mm0, <mem_A> | // [A41*X1, A31*X1, A21*X1, A11*X1] |
| MOVD | mm1, <mem_X + 4> | // [0, 0, 0, X2] |
| DUPLS | mm1, mm1 | // [0, 0, X2, X2] |
| DUPLD | mm1, mm1 | // [X2, X2, X2, X2] |
| PFMUL | mm1, <mem_A + 16> | // [A42*X2, A32*X2, A22*X2, A12*X2] |
| MOVD | mm2, <mem_X + 8> | // [0, 0, 0, X3] |
| DUPLS | mm2, mm2 | // [0, 0, X3, X3] |
| DUPLD | mm2, mm2 | // [X3, X3, X3, X3] |
| PFMUL | mm2, <mem_A + 32> | // [A43*X3, A33*X3, A23*X3, A13*X3] |
| MOVD | mm3, <mem_X + 12> | // [0, 0, 0, X4] |
| DUPLS | mm3, mm3 | // [0, 0, X4, X4] |
| DUPLD | mm3, mm3 | // [X4, X4, X4, X4] |
| PFMUL | mm3, <mem_A + 48> | // [A44*X4, A34*X4, A24*X4, A14*X4] |
| PFADD | mm0, mm1 | // [A42*X2 + A41*X1, A32*X2 + A31*X1, A22*X2 + A21*X1, A12*X2 + A11*X1] |
| PFADD | mm2, mm3 | // [A44*X4 + A43*X3, A34*X4 + A33*X3, A24*X4 + A23*X3, A14*X4 + A13*X3] |

US 6,418,529 B1

3

4

TABLE 1-continued

| | | |
|-------|--------------|---|
| PFADD | mm0, mm2 | // [A44*X4 + A43*X3 + A42*X2 + A41*X1, // A34*X4 + A33*X3 + A32*X2 + A31*X1, // A24*X4 + A23*X3 + A22*X2 + A21*X1, // A14*X4 + A13*X3 + A12*X2 + A11*X1] |
| MOVDQ | <mem_Y>, mm0 | // store [Y4, Y3, Y2, Y1] |

Accordingly, there is a need in the technology for providing an apparatus and method which efficiently performs multi-dimensional computations based on a “horizontal” or intra-add operation. There is also a need in the technology for a method and operation for increasing code density by eliminating the need for the rearrangement of data elements and the corresponding rearrangement operations.

BRIEF SUMMARY OF THE INVENTION

A method and apparatus for including in a processor instructions for performing intra-add operations on packed data is described. In one embodiment, an execution unit is coupled to a storage area. The storage area has stored therein a first and a second packed data operands. The execution unit performs operations on data elements in the first and the second packed data operands to generate a plurality of data elements in a packed data result in response to receiving a single instruction. At least two of the plurality of data elements in the packed data result store the result of an intra-add operation upon the first and the second packed data operands.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated by way of example, and not limitation, in the figures. Like reference indicate similar elements.

FIG. 1 illustrates the vertical or inter-add operation of the prior art.

FIG. 2 illustrates the horizontal or intra-add operation in accordance with one embodiment of the present invention.

FIG. 3A–3C illustrate matrix multiplication based on horizontal or intra-add operations in accordance with one embodiment of the present invention.

FIG. 4A–4G illustrate matrix multiplication based on horizontal or intra-add operations in accordance with another embodiment of the present invention.

FIG. 5 illustrates an exemplary computer system in accordance with one embodiment of the invention.

FIG. 6 illustrates packed data-types in accordance with one embodiment of the invention.

FIG. 7 illustrates one embodiment of a floating point packed data format.

FIG. 8 is a flow diagram illustrating a method for performing the intra-add operation of FIG. 2 in accordance with one embodiment of the invention.

FIG. 9 illustrates a circuit for performing the intra-add operation of FIG. 2 in accordance with one embodiment of the invention.

FIG. 10 is a general block diagram illustrating the use of matrix multiplication based on horizontal or inter-add operations, in rendering graphical objects in animation.

DETAILED DESCRIPTION OF THE INVENTION

In the following description, numerous specific details are set forth to provide a thorough understanding of the inven-

tion. However, it is understood that the invention may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in detail in order not to obscure the invention.

OVERVIEW

One aspect of the present invention is a processor including instructions for performing horizontal or intra-addition operations on packed data. The horizontal add or intra-add operation adds together pairs of data elements in a packed data operand. In one embodiment, two pairs of data elements (e.g., X_3 and X_2 , and X_1 and X_0) located within a single storage area (e.g., Source1) are added together using a horizontal or an intra-add operation. In an alternate embodiment, data elements from each of two storage areas (e.g., Source1 and Source2) are added and stored as data elements of a resulting packed data, as shown in FIG. 2.

Another aspect of the present invention involves a method and apparatus for performing matrix multiplication using a horizontal or intra-addition operation. In one embodiment, each 32-bit data element from a 1x2 vector is multiplied with corresponding 32-bit data elements from each row of a 2x2 matrix, as shown in FIGS. 3A and 3B, generating two pairs of 64-bit intermediate results, each pair of which are stored in separate storage areas. An intra-add operation is performed on each pair of the intermediate results to generate a pair of data elements, which are stored as a packed result, as shown in FIG. 3C. The example on FIGS. 3A–3C illustrate the application of one embodiment of the present invention using an integer data format in which the full width of the result is stored in a result register. In alternative embodiments, the result register is the same size as the source register.

The operation of a further example of a matrix multiplication operation based on intra-add operations is shown in FIGS. 4A through 4G, and is representative of a multiplication operation between a 4x4 matrix and a 4x1 vector (such as the multiplication of a 4x4 matrix A with a 4x1 vector X to provide a 4x1 vector Y as described earlier). In particular, each data element from a 4x1 vector is multiplied with the corresponding data elements from each row of a 4x4 matrix, as shown in FIGS. 4A through 4D.

A first pair of intra-add operations are then performed on the initial resulting data elements (IResult1+IResult2, IResult3+IResult4), as shown in FIG. 4E and 4F; followed by a second single intra-add operation on the results of the first intra-add operations (IAResult1+IAResult2), to obtain the final result (Result) as shown in FIG. 4G.

Although the examples illustrated in FIGS. 4A–4G are floating point examples, the present invention may also be applied to packed integer data. Matrix multiplication based on horizontal add or intra-add operations only requires 12 instructions, as compared to the typical 20 instructions required when the same matrix multiplication is performed using vertical add or inter-add operations. Exemplary code for implementing matrix multiplication based on horizontal add or intra-add operations is shown in Table 2:

US 6,418,529 B1

5

Exemplary Code Based on Horizontal-Add
Operations

PFADDM represents the Intra-add instruction of the present invention.

Assumptions:

TABLE 2

| | | |
|--|-------------------|--|
| 1/X stored with X1 first, X2 second, . . . , X4 last | | |
| 2/A stored with A11 first, A12 second, A13 third, etc. | | |
| MOVDQ | mm0, <mem_X> | // [X4, X3, X2, X1] |
| MOVDQ | mm3, mm0 | |
| PFMUL | mm0, <mem_A> | // [A14*X4, A13*X3, A12*X2, A11*X1] |
| MOVDQ | mm1, mm3 | |
| PFMUL | mm1, <mem_A + 16> | // [A24*X4, A23*X3, A22*X2, A21*X1] |
| MOVDQ | mm2, mm3 | |
| PFMUL | mm2, <mem_A + 32> | // [A34*X4, A33*X3, A32*X2, A31*X1] |
| PFMUL | mm3, <mem_A + 48> | // [A44*X4, A43*X3, A42*X2, A41*X1] |
| PFADDM | mm0, mm1 | // [A24*X4 + A23*X3, A22*X2 + A21*X1, // A14*X4 + A13*X3, A12*X2 + A11*X1] |
| PFADDM | mm2, mm3 | // [A44*X4 + A43*X3, A42*X2 + A41*X1, // A34*X4 + A33*X3, A32*X2 + A31*X1] |
| PFADDM | mm0, mm2 | // [A44*X4 + A43*X3 + A42*X2 + A41*X1, // A34*X4 + A33*X3 + A32*X2 + A31*X1], // A24*X4 + A23*X3 + A22*X2 + A21*X1, // A14*X4 + A13*X3 + A12*X2 + A11*X1] |
| MOVDQ | <mem_Y>, mm0 | // store [Y4, Y3, Y2, Y1] |

6

processors, execution unit **120** recognizes instructions in packed instruction set **122** for performing operations on packed data formats. Packed instruction set **122** includes instructions for supporting intra-add and multiply operations. In addition, packed instruction set **122** may also include other packed instructions.

Although the discussions above pertain to a horizontal-add or intra-add instruction, alternative embodiments could in addition to, or in place of the intra-add instruction, have an intra-subtract instruction or element operation. In this case, one of a pair of data elements within a packed data will be subtracted from a second element of the pair of data elements to accomplish the intra-subtract operations.

In addition, although the discussions above pertain to packed operands that have four data elements, alternative embodiments may involve packed operands that have at least two data elements (i.e., that are double wide).

COMPUTER SYSTEM

FIG. 5 illustrates one embodiment of a computer system **100** which implements the principles of the present invention. Computer system **100** comprises a bus **102** for communicating information, and a processor **110** for processing information. In one embodiment, the bus **102** may be any communications hardware and/or software for communicating information. The processor **110** represents a central processing unit of any type of architecture, examples of which include a CISC, a RISC or a VLIW type architecture. Computer system **100** further comprises a main memory **104** that is coupled to bus **102**, for storing information and instructions to be executed by the processor **110**. Computer system **110** also comprises a read only memory (ROM) **106** and/or other status storage device, coupled to the bus **102** for storing information and instructions for access and execution by processor **110**. In addition, computer system **110** comprises a data storage device **108** that is coupled to the bus **102** for storing information and instructions.

As shown in FIG. 5, processor **110** comprises an execution unit **120**, a set of register file(s) **130**, a cache memory **140**, a decoder **150** and an internal bus **160**. The processor **110** also includes additional circuitry (not shown) which is not necessary to the understanding of the present invention.

Execution unit **120** is used for executing instructions received by processor **110**. In addition to recognizing instructions typically implemented in general purpose

Execution unit **120** is coupled to register file **130** by internal bus **160**. Register file **130** represents a storage area on processor **110** for storing information, including data. It is understood that the aspects of the invention are the described intra-add instruction set and a code sequence for performing matrix multiplication for operating on packed data. According to these aspects of the invention, the storage area used for storing the packed data is not critical. Execution unit **120** is coupled to cache **140** and decoder **150**. Cache **140** is used to cache data and/or control signals (such as instructions) from, for example, main memory **104**. Decoder **150** is used for decoding instructions received by processor **110** into control signals and/or microcode entry points. In response to these control signals and/or microcode entry points, execution unit **120** performs the appropriate operations. Decoder **150** may be implemented using any number of different mechanisms (e.g., a look-up table, a hardware implementation, a PLA, etc.).

FIG. 5 additionally shows a data storage device **108**, (e.g., a magnetic disk, optical disk, and/or other machine readable media) coupled to computer system **100**. In addition, the data storage device **108** is shown including code **195** for execution by the processor **110**. The code **195** can be written to cause the processor **110** to perform matrix multiplication with the intra-add instruction for any number of purposes (e.g., 3-D graphics multiplication, 3-D transformation, 3-D rotation, 3-D scaling, etc.). Computer system **100** can also be coupled via bus **102** to a display device **170**, a user input device **172**, a hard copy device **176**, a sound recording and/or playback device **178**, a video digitizing device **180**, and/or a communications device **190** (e.g., a serial communications chip, an ethernet chip or a modem, which provides communications with an external device or network).

DATA AND STORAGE FORMATS

Generally, a data element is an individual piece of data that is stored in a single register (or memory location) with other data elements of the same length. The number of data elements stored in a register is the number of bits supported by the packed data format (e.g., 64 bits for integer packed

US 6,418,529 B1

7

data) divided by the length in bits of a data element. While any number of packed data formats can be used, FIGS. 6–7, respectively, illustrate integer and floating-point packed data-types according to one embodiment of the invention.

Three integer packed data formats are illustrated in FIG. 6: packed byte **401**, packed word **402**, and packed doubleword **403**. While in one embodiment, each of the packed data formats in FIG. 6 can be either signed or unsigned formats, alternative embodiments support only signed or unsigned formats. Packed byte **401**, in one embodiment of the invention, is sixty-four bits long containing eight data elements. Each data element is one byte long. Packed word **402** is sixty-four bits long and contains four word **402** data elements. Each word **402** data element contains sixteen bits of information. Packed doubleword **403** is sixty-four bits long and contains two doubleword **403** data elements. Each doubleword **403** data element contains thirty-two bits of information.

FIG. 7 shows one floating point packed data format having four 32-bit data elements. While one floating point packed data format is illustrated, alternative embodiments could support a different and/or additional floating point packed data formats.

INTRA-ADD OPERATION(S)

In one embodiment of the invention, the SRC1 register contains packed data (Source1), the SRC2 register contains packed data (Source2) and the DEST register will contain the result (Result) of performing the horizontal add instruction on Source1 and Source2. In the first step of the horizontal add instruction, one or more pairs of data elements from Source1 are summed together. Similarly, one or more pairs of data elements from Source2 are summed together. The results of the instruction are then stored in the DEST register.

FIG. 8 is a flow diagram illustrating a process **S800** for performing the intra-add operation of FIG. 2 according to one embodiment of the present invention. The process **S800** begins from a start state and proceeds to process step **S802**, where the decoder **150** decodes the control signal received by processor **110**. In particular, the decoder **150** decodes the operation code for the intra-add instruction.

The process **S800** then advances to process step **S804**, where the device **150** accesses registers in register file **130** given the SRC1 **602** and SRC2 **603** addresses. Register file **130** provides the execution unit **120** with the packed data stored in the SRC1 **602** register (Source1), and the packed data stored in SRC2 **603** register (Source2).

The process **S800** proceeds to process step **S806**, where the decoder **150** enables the execution unit **120** to perform the instruction. Next, the process **S800** performs the following series of steps, as shown in process step **S808** and FIG. 2. Source1 bits thirty-one through zero are added to Source1 bits sixty-three through thirty-two, generating a first 32-bit result (Result[31:0]). Source1 bits ninety-five through sixty-four are added to Source1 bits one hundred-and-twenty-seven through ninety-six, generating a second 32-bit result (Result[63:32]). Source2 bits thirty-one through zero are added to Source2 bits sixty-three through thirty-two, generating a first 32-bit result (Result[95:64]). Source2 bits ninety-five through sixty-four are added to Source1 bits one hundred-and-twenty-seven through ninety-six, generating a second 32-bit result (Result[127:96]).

The process **S800** advances to process step **S810**, where the results of the intra-add instruction are stored in DEST. The process **S800** then terminates. Of course, the method of

8

FIG. 8 can be easily altered to describe the horizontal addition of other packed data formats.

EXEMPLARY INTRA-ADD CIRCUIT

In one embodiment, the intra-add instructions can execute on multiple data elements in the same number of clock cycles as an inter-add operation on unpacked data. To achieve execution in the same number of clock cycles, parallelism is used.

FIG. 9 illustrates a circuit **900** for performing intra-add operation of FIG. 2 according to one embodiment of the invention. Operation control **910** processes the control signal for the intra-add operations. Operation control **910** outputs signals via signal line **920** to control intra-adder **930**.

The intra-adder **930** receives inputs from Source1[127:0], Source2[127:0], and Enable **920**. The intra-adder **930** includes four adder circuits **932**, **934**, **936** and **938**. Adder **932** receives inputs from Source2[127:64], adder **934** receives inputs from Source2[63:0], adder **936** receives inputs from Source1[127:64], while adder **938** receives inputs from Source1[63:0]. When enabled, the adders **932**, **934**, **936** and **938** sum their respective inputs, and each generates a 32-bit output. The results of the addition by adder **932** (i.e., Result[127:96]), adder **934** (i.e., Result[95:64]), by adder **936** (i.e., Result[63:32]), and by adder **938** (i.e., Result[31:0]) are combined into the 128-bit Result and communicated to the Result Register **940**.

MATRIX MULTIPLICATION USING INTRA-ADD OPERATION(S)

FIG. 10 is a general block diagram illustrating the use of matrix multiplication based on a horizontal or intra-add operation for rendering graphical objects in animation according to one embodiment of the invention. FIG. 10 shows a computer system **1000** containing digital data **1010** representing 3-dimensional (3D) graphics. The digital data **1010** may be read from a storage medium or generated real time. At sometime, the conversion unit **1020** alters data using 3D geometry (e.g., by performing a 3D transformation) through the implementation of matrix multiplication based on a horizontal add operation to rotate a 3D object in providing animation. The resulting graphical object **1030** (e.g., see FIGS. 4A–4G) is then displayed on a screen display **1040** using well known techniques. While FIG. 10 shows that the resulting graphical object **1030** is displayed, the resulting graphical object may alternatively be stored, transmitted, etc.

In one embodiment, the computer system **100** shown in FIG. 5 is used to implement the computer system **1000** from FIG. 10. In this embodiment, the digital data **1010** from FIG. 10 is any data stored in the storage device **110** representing 3D graphics. In one embodiment, the conversion unit **1020** from FIG. 8 is implemented using the processor **110** and the code **195** to alter data using 3D geometry. For example, data is altered to perform a transformation. In this embodiment, the processor **110**, executing the code **195**, performs the transformation and stores the transformed data **1030** for display.

CONCLUSION

The intra-add operation facilitates the efficient performance of multi-dimensional computations. It further increases code density by eliminating the need for the rearrangement of data elements and the corresponding rearrangement operations.

US 6,418,529 B1

9

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended 5 claims rather than the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed is:

1. A processor comprising:

a storage area to store a first packed data operand and a second packed data operand; and

an execution unit coupled to said storage area, the execution unit in response to receiving a single instruction to perform operations on data elements in said first 15 Packed data operand and said second packed data operand to generate a plurality of data elements in a packed data result, at least two of said plurality of data elements in said packed data result being the result of an intra-add operation performed by the execution unit 20 using one of said first packed data operand and said second packed data operand.

2. The processor of claim 1, wherein

said first packed data operand and said second packed data operand each include a first pair of data elements; 25

one of the at least two of said plurality of data elements in said packed data result respectively represents the result of the intra-add operation performed by the execution unit adding together the first pair of data 30 elements in the first packed data operand; and

another one of the at least two of said plurality of data elements in said packed data result respectively represents the result of the intra-add operation performed by the execution unit adding together the first pair of data 35 elements in the second packed data operand.

3. The processor of claim 2 wherein

the execution unit performs a first intra-add operation on the first packed data operand and a second intra-add operation on the second packed data operand. 40

4. The processor of claim 1, wherein the at least two of said plurality of data elements in said first and second packed data operands and in said packed data result are floating point data.

5. The processor of claim 1, wherein

said first packed data operand and said second packed data operand each include two pairs of data elements; and

said packed data result has a first pair of data elements to represent the result of the execution unit summing each pair of data elements of the two pair of data elements in the first packed data operand; and 50

said packed data result has a second pair of data elements to represent the result of the execution unit summing each pair of data elements of the two pair of data elements in the second packed data operand. 55

6. The processor of claim 5, wherein

the pairs of data elements are floating point data.

7. The processor of claim 6, wherein each of the pair of 60 data elements has the same number of bits.

8. The apparatus of claim 1, wherein said data elements are either unsigned or signed data.

9. The processor of claim 1 wherein

operations performed by the execution unit on the data 65 elements in said first and second packed data operands further includes one or more of the set of arithmetic

10

operations of inter-add, inter-subtract, or inter-multiply between data elements of said first and second packed data operands.

10. The processor of claim 1 wherein

operations performed by the execution unit on the data elements in said first and second packed data operands further includes one or more arithmetic operations between data elements of the set of intra-add, intra-subtract, or intra-multiply on data elements within said first packed data operand and data elements within said second packed data operand.

11. The processor of claim 1 wherein

the intra-add operation performed by the execution unit on the first packed data operand or the second packed data operand is an addition operation amongst data elements within the same packed data operand.

12. A processor comprising:

a first storage area for storing a first packed data operand, containing at least an A data element and a B data element packed together;

a second storage area for storing a second packed data operand containing at least a C data element and a D data element packed together;

an add circuit responsive to execution of a single instruction the add circuit including

a first adder coupled to said first storage area to add said A and said B data elements together generating an output result in response to execution of a single instruction, and

a second adder coupled to said second storage area to add said C and said D data elements together generating an output result in response to the single instruction; and

a third storage area coupled to said first and said second adders, said third storage area having at least a first field and a second field, said first field for storing the output result of said first adder as a first data element of a third packed data, said second field for storing the output result of said second adder as a third data element of said third packed data.

13. The processor of claim 12, wherein

said first packed data operand further contains an E data element and an F data element;

said second packed data operand further contains a G data element and an H data element;

said add circuit further including

a third adder coupled to said first storage area to add said E and said F data elements to generate an output result, and

a fourth adder coupled to said second storage area to add said G and said H data elements to generate an output result; and

said third storage area further coupled to said third and said fourth adders, said third storage area further having a third field and a fourth field, said third field for storing the output result of the third adder as a second data element of said third packed data, said fourth field for storing the output result of the fourth adder as a fourth data element of said third packed data.

14. The processor of claim 13, wherein said A, B, C, D, E, F, G and H data elements are either signed or unsigned data.

15. The processor of claim 13, wherein

to add said A and B data elements together is a first intra-add operation,

US 6,418,529 B1

11

to add said C and D data elements together is a second intra-add operation,

to add said E and F data elements together is a third intra-add operation, and

to add said G and H data elements together is a fourth intra-add operation. 5

16. The processor of claim 12, wherein the data elements are floating point data.

17. The processor of claim 12, wherein each data element has the same number of bits. 10

18. The processor of claim 12, wherein said data elements are either unsigned or signed data.

19. The processor of claim 12, wherein

to add said A and B data elements together is a first intra-add operation, and 15

to add said C and D data elements together is a second intra-add operation.

20. A processor comprising:

a decoder configured to decode instructions, and 20

a circuit coupled to said decoder, said circuit in response to a single decoded instruction being configured to:

add together data elements in a first packed data operand to generate a first result data element;

add together data elements in a second packed data operand to generate a second result data element; and 25

store said first and second result data elements in a register for use as a packed data operand.

21. The processor of claim 20, wherein said data elements in said first packed data operand, said second packed data operand and said packed data operand are floating point data. 30

22. The processor of claim 20, wherein each data element has the same number of bits.

23. The processor of claim 20 further comprising: 35

a first source register to store said first packed data operand, and

a second source register to store said second packed data operand. 40

24. The processor of claim 20, wherein

to add data elements together in the first packed data operand is a first intra-add operation, and

to add data elements together in the second packed data operand is a second intra-add operation. 45

25. A method of single instruction execution in a computer, the method comprising:

a) decoding a single instruction;

b) in response to said decoding of the single instruction, adding a first value and a second value of a first packed data operand together to generate a first data element; and 50

c) completing execution of said single instruction by storing said first data element in a first field of a storage area. 55

26. The method of claim 25, further comprising:

adding a third value and a fourth value of the first packed data operand together to generate a second data element, and 60

completing execution of said single instruction by storing said second data element in a second field of the storage area.

27. The method of claim 26, wherein

the adding together of the first value and the second value of the first packed data operand is a first intra-add operation, and 65

12

the adding together of the third value and the fourth value of the first packed data operand is a second intra-add operation.

28. The method of claim 25, further comprising:

prior to adding the first value and the second value of the first packed data operand together,

storing the first packed data operand in the storage area.

29. The method of claim 25, wherein

the adding together of the first value and the second value of the first packed data operand is a first intra-add operation.

30. A method for manipulating a first packed data and a second packed data in a computer system, the first packed data having an A data element and a B data element adjacent to the A data element, the second packed data having a C data element and a D data element adjacent to the C data element, the method comprising

a) decoding a single instruction;

b) in response to said decoding of said single instruction, performing the operation of (A+B) to generate a first data element in a third packed data, and performing the operation of (C+D) to generate a second data element in the third packed data; and

c) storing said third packed data in a storage area.

31. The method of claim 30, further comprising:

prior to performing the operation of (A+B) and the operation of (C+D),

storing the first packed data into the storage area, and

storing the second packed data into the storage area.

32. The method of claim 30, wherein

the performance of the operation of (A+B) is a first intra-add operation, and

the performance of the operation of (C+D) is a second intra-add operation.

33. The method of claim 30, wherein

the first packed data further has an E data element and an F data element adjacent the E data element,

the second packed data further has a G data element and an H data element adjacent the G data element,

and in response to said decoding of said single instruction, prior to said storing of said third packed data in the storage area, the method further includes

performing the operation of (E+F) to generate a third data element in the third packed data, and

performing the operation of (G+H) to generate a fourth data element in the third packed data.

34. The method of claim 33, further comprising:

prior to performing the operations,

storing the first packed data into the storage area, and

storing the second packed data into the storage area.

35. The method of claim 33, wherein

the performance of the operation of (A+B) is a first intra-add operation,

the performance of the operation of (C+D) is a second intra-add operation,

the performance of the operation of (E+F) is a third intra-add operation, and

the performance of the operation of (G+H) is a fourth intra-add operation.

36. A method of executing a single instruction in a computer, said method comprising:

summing a pair of data elements within a first packed data operand of said single instruction;

summing a pair of data elements within a second packed data operand of said single instruction; and

US 6,418,529 B1

13

storing the summed pairs as separate data elements in a result packed data operand for use by another instruction.

37. The method of claim 36, wherein said data elements are floating point data. 5

38. The method of claim 36, further comprising:
prior to summing the pair of data elements in the first packed data operand and the second packed data operand,
storing the first packed data operand into a first register, 10
and
storing the second packed data into a second register.

39. The method of claim 36, wherein
the summing of the pair of data elements within the first packed data operand is a first intra-add operation, and 15
the summing of the pair of data elements within the second packed data operand is a second intra-add operation.

40. A method for matrix multiplication in a processor, the method comprising: 20
responsive to the processor receiving a single instruction,
multiplying each data element of a first row of a matrix source to each respective data element of a vector source to generate a plurality of data elements of a first intermediate result, 25
multiplying each data element of a second row of the matrix source to each respective data element of the vector source to generate a plurality of data elements of a second intermediate result,
performing an intra-addition operation on the plurality of data elements of the first intermediate result to generate a first data element of a packed result, and
performing an intra-addition operation on the plurality of data elements of the second intermediate result to generate a second data element of the packed result. 35

41. The method of claim 40 wherein the matrix source is a two by two matrix and the vector source is a two by one matrix and the packed result is a two by one matrix.

42. The method of claim 40 wherein 40
the intra-addition operation of the first intermediate result are horizontal addition operations amongst the plurality of data elements in the first intermediate result, and
the intra-addition operation of the second intermediate result are horizontal addition operations amongst the plurality of data elements in the second intermediate result. 45

43. The method of claim 40 wherein
the first intermediate result is a packed data operand having the plurality of data elements, and 50
the second intermediate result is a packed data operand having the plurality of data elements.

44. A method for matrix multiplication in a processor, the method comprising: 55
responsive to the processor receiving a single instruction,
multiplying each data element of a first row of a matrix source to each respective data element of a vector source to generate a plurality of data elements of a first intermediate result,
multiplying each data element of a second row of the matrix source to each respective data element of the vector source to generate a plurality of data elements of a second intermediate result, 60
multiplying each data element of a third row of the matrix source to each respective data element of the vector source to generate a plurality of data elements of a third intermediate result, 65

14

multiplying each data element of a fourth row of the matrix source to each respective data element of the vector source to generate a plurality of data elements of a fourth intermediate result,

performing an intra-addition operation on the plurality of data elements of the first intermediate result to generate first and second data elements of a first intra-add result,

performing an intra-addition operation on the plurality of data elements of the second intermediate result to generate third and fourth data elements of the first intra-add result,

performing an intra-addition operation on the plurality of data elements of the third intermediate result to generate first and second data elements of a second intra-add result,

performing an intra-addition operation on the plurality of data elements of the fourth intermediate result to generate third and fourth data elements of the second intra-add result,

performing an intra-addition operation on data elements of the first intra-add result to generate first and second data elements of a packed output result, and
performing an intra-addition operation on data elements of the second intra-add result to generate third and fourth data elements of the packed output result.

45. The method of claim 44 wherein the matrix source is a four by four matrix and the vector source is a four by one matrix and the packed output result is a four by one matrix.

46. The method of claim 44 wherein
the intra-addition operation of the first intermediate result are horizontal addition operations amongst the plurality of data elements in the first intermediate result,
the intra-addition operation of the second intermediate result are horizontal addition operations amongst the plurality of data elements in the second intermediate result, 40

the intra-addition operation of the third intermediate result are horizontal addition operations amongst the plurality of data elements in the third intermediate result, and
the intra-addition operation of the fourth intermediate result are horizontal addition operations amongst the plurality of data elements in the fourth intermediate result.

47. The method of claim 44 wherein
the first intermediate result is a packed data operand having the plurality of data elements,
the second intermediate result is a packed data operand having the plurality of data elements,
the third intermediate result is a packed data operand having the plurality of data elements, and
the fourth intermediate result is a packed data operand having the plurality of data elements.

48. A method in a processor for executing an intra-add operation using operands, the method comprising:
responsive to the processor receiving a single intra-add instruction and a first and second operand, each of the first and second operand being a packed data type and having a plurality of data elements of an even number, the even number of the plurality of data elements forming pairs of data elements from one end to another end of each operand,
for the first operand, summing the data elements of each pair of data elements together to generate lower order data elements of a resultant, and
for the second operand, summing the data elements of each pair of data elements together to generate

US 6,418,529 B1

15

higher order data elements of the resultant, the resultant having a plurality of data elements of the even number.

49. The method of claim 48, wherein each of the data elements represent floating point data. 5
50. The method of claim 48, wherein each of the data elements has the same number of bits.
51. The method of claim 48, wherein each of the data elements represents unsigned or signed data. 10
52. A processor responsive to a single instruction having a first packed data operand, the processor comprising: 15
- means for decoding the single instruction;
 - means for summing a pair of data elements within the first packed data operand of the single instruction in response thereto; and 20
 - means for storing the summed pair as a separate data element in a packed result operand for use by another instruction. 25
53. The processor of claim 52 further comprising:
- means for performing one or more operations of the set of arithmetic operations of inter-add, inter-subtract, or inter-multiply between respective data elements of the first packed data operand and a second packed data operand. 30
54. The processor of claim 52 further comprising:
- means for performing one or more operations of the set of arithmetic operations between data elements of the set of intra-add, intra-subtract, or intra-multiply on data elements within the first packed data operand and data elements within a second packed data operand. 35

16

55. The processor of claim 52, further comprising: a storage means to store the first packed data operand.
56. The processor of claim 52 wherein, the means for summing the pair of data elements within the first packed data operand is a means to intra-add.
57. A processor comprising: 40
- a storage area to store a first packed data operand having data elements; and
 - an execution unit coupled to the storage area, the execution unit in response to receiving a single instruction to perform operations on the data elements in the first packed data operand to generate a plurality of data elements in a packed data result, at least one of the plurality of data elements in the packed data result being the result of an intra-add operation performed by the execution unit using at least two of the data elements in the first packed data operand. 45
58. The processor of claim 57, wherein 50
- the storage area to store a second packed data operand having data elements; and
 - the execution unit in response to receiving the single instruction to perform operations on the data elements in the second packed data operand to generate a plurality of data elements in the packed data result, at least another one of the plurality of data elements in the packed data result being the result of an intra-add operation performed by the execution unit using at least two of the data elements in the second packed data operand. 55

* * * * *